CR-180709

MCR 85-594

FINAL REPORT

SATELLITE OPERATIONS SUPPORT EXPERT SYSTEM

CONTRACT NAS5-28038

May 13, 1985

Prepared For:

NASA - Goddard Space Flight Center
Greenbelt, Maryland

MARTIN MARIETTA DENVER AEROSPACE
P.O. BOX 179
DENVER, CO 80201

TABLE OF CONENTS

I. INTRODUCTION

The Satellite Operations Support Expert System contract is an effort to identify aspects of satellite ground support activity at Goddard Space Flight Center (GSFC) which could profitably be automated with artificial intelligence (AI) and to develop a feasibility demonstration for the automation of one such area. To accommodate the one-year contract time frame, GSFC selected the International Sun-Earth Explorer (ISEE) control facility as the application domain. The three ISEE satellites are relatively simple so that operational intricacies were expected to not complicate the development process. This allowed the resultant system to illustrate the application and development of AI technology in the satellite support environment.

The importance of considering automation in general is addressed at several points in this document. Automation can be considered as a means for potentially increasing the effectiveness with which a task is performed and decreasing the long run costs. This essential cost-benefit tradeoff will be considered later in this document.

If automation is considered, then it may be profitable to examine the possible use of artificial intelligence technologies. In many instances, the decision is quite straightforward. Many applications--though perhaps rather complex--can be suitably automated with conventional hardware and software. In cases where the task is more intricate, involves substantial experiential, non-textbook knowledge, or is perhaps less constrained, expert reasoning tools from AI may prove to be appropriate.

Since the focus of this contract was on the illustration of AI technology and system development, the above considerations were traded off in favor of providing insight into the workings of expert systems. As explained in greater detail later, we decided to use the hydrazine propulsion subsystems (HPS) of the International Sun-Earth Explorer (ISEE) and the International Ultraviolet Explorer (IUE) as application domains. This was after extensive examination of candidate applications and the realization that all of them, save one or two at the most abstract research-oriented level, could be more efficiently automated using conventional approaches.

We chose to build a demonstration fault handling system for an HPS. The system is written in Franz Lisp and is currently hosted on a VAX 11/750-11/780 family machine. As detailed in a later section, it allows the user to select which HPS (either from ISEE or from IUE) to run the system on. The user then chooses the fault desired for the run. The demonstration system generates telemetry corresponding to the particular fault. The completely separate fault handling module then uses this telemetry to determine what and where the fault is and how to work around it. Graphics are used to depict the structure of the HPS, and the telemetry values displayed on the screen are continually updated. The user has the option of having each rule displayed as it is fired and of having the system describe its actions and reasoning at each step. The system can be used to handle up to two sequential faults to demonstrate the fault handling process for spacecraft in degraded conditions.

This final report describes the capabilities of this system and its development cycle. The following sections discuss:

II - The choice of application domain
III - The process of requirements analysis
IV - The design process
V - The implementation experience
VI - Advantages of an approach to automation
VII - Results of the contract experience
VIII - Conclusions drawn from the experience as to the nature and appropriateness of automation

A user's manual for the demonstration fault handling system has been prepared under separate cover.

II. CHOICE OF THE APPLICATION DOMAIN

In the first task of this contract, we sought to identify activity areas within the GSFC control center that could benefit from automation via artificial intelligence (AI). We found this a difficult task: while many areas could benefit from standard automation, few appeared intricate enough to justify the use of AI. An important step in the analysis of possible applications for AI techniques was the decomposition of activities within the ISEE control center into six categories:

1) telemetry monitoring (for both experiments and spacecraft operations);
2) deciding what action to take in response to known problems;
3) deciding what action to take for novel problems;
4) ground coverage scheduling;
5) experiment scheduling; and
6) experiment management.

A detailed discussion of the automation of each of these activities is included in Appendix D of this report. To summarize, although most activities needed automation via conventional software, two activities appeared to be complex enough to use AI. These were command generation and procedure generation. The first of these is the process of identifying a set of commands which fulfill the request of an experimenter. An experimenter might request, for example, that a sensor be focused on a particular stellar object. An automated command generation system would determine what needed to be done on the spacecraft to effect the request--the spacecraft may need to be rotated, other experiments may need to be modified or turned off, etc. The system would then generate a series of commands that could be uplinked to the satellite to make it do what was needed. A procedure generation system would do the same type of processing but for spacecraft operations rather than the onboard experiments.

Although the initial target domain was ISEE, it appeared that it would be useful to also consider the more complex control center for IUE. After further analysis it was concluded that for ISEE automated command generation was easily implemented with conventional software because of the limited set of commands available and their applicability to only one procedure. This led to consideration of procedure generation for spacecraft management. Martin Marietta personnel with experience in spacecraft support and housekeeping systems identified a number of problems which were examined during another on-site visit to the ISEE control center. However, it became clear that the complexity of procedure generation was lacking in the ISEE context. The procedures in ISEE are a set of unrelated activities to be performed at a specific time. For comparison, procedure generation for IUE was analyzed. The procedures in the IUE context were more complex than those in ISEE and each procedure had a logical coherence. Unfortunately, as with ISEE, few procedures were conceptually connected to other procedures.

If this interconnection existed, then a general body of knowledge about a particular procedural area would be used by various logical processes to generate a number of different procedures. This would illustrate the function and interaction of different reasoning processes in an expert system. Since this interconnection did not exist, a specific set of information and a specific, different set of rules would be used to generate each procedure. Few, if any, rules could be used in the generation of more than one procedure. As a result, it was decided that neither command generation or procedure generation was an appropriate activity for illustrating the various aspects of expert systems.

As discussed in the introduction to the requirements document (see Appendix E of this report), three additional possibilities for using expert systems techniques emerged: 1) in executive control functions, 2) in high-level mission workarounds, and 3) for fault handling. The first two were eventually dismissed, partly because automation considerations in these areas should be incorporated in the design phase of satellite system development.

Since the intent of this effort was to provide an example of artificial intelligence approaches in the satellite control domain, it was decided that fault handling could be a suitable application. The hydrazine propulsion subsystem (HPS) was selected as the focus for this application for several reasons, as discussed in the requirements document. Chief among these is that although quite straightforward, the HPS is nonetheless one of the most complex subsystems on ISEE. A fault handling system for an HPS could illustrate the AI concepts of rule-based reasoning and representation while being simple enough not to be obscured with technical minutia.

It was also noted that there exists considerable similarity between HPSs on different spacecraft. It was reasoned that it would be relatively easy to generalize the diagnostic process and thus be able to operate the demonstration system on multiple HPS configurations. Although this was desirable, it was not clear that it would be possible to build the simulation system such that it could generate the telemetry data necessary to support this within the scope of the contract. However, it was decided to attempt this since it would illustrate the flexibility achievable in automated reasoning systems.

III. REQUIREMENTS ANALYSIS

A portion of this phase was spent understanding hydrazine propulsion systems in detail. This information is documented in Section II of Appendix E. From this exploration, we established the set of faults which the demonstration system would handle. These are:

- Leak in the hydrazine storage tank
- Leak in a section of the line
- Failure of a latch valve to open
- Failure of a latch valve to close
- Failure of an engine valve to open
- Failure of an engine valve to close
- Failure of a heater to turn off
- Failure of a heater to turn on
- Failure of a catalyst bed

In addition to these, we originally believed that clogged filters would be an appropriate fault. Further exploration, however, established that there is no way to distinguish between a completely clogged filter and a valve that is stuck closed. Since these faults have identical symptoms, we decided to exhibit only one, the stuck valve, for the purpose of the demonstration. Also, we decided to include certain sensor failures as possible faults. These are failures in engine valve and latch valve sensors (which read open or closed), fuel pressure sensors, and fuel line and catalyst bed temperature sensors (which read a specific numeric value).

Section III of the requirements document details the symptoms associated with each (non-sensor) fault and the appropriate workaround. These, with the above modifications, are the primary specifications to which the fault handling portion of the demonstration system was built.

After the initial requirements analysis, it was decided to design more complexity into the demonstration system. Thus, we chose to build the system to be able to handle faults for multiple HPS configurations (within logical limits). It was decided to have the ISEE and IUE configurations available and to allow the user to design his own configuration as well. (For time and simulation complexity considerations, the user configuration option was later deleted. Although untested, the diagnostic code is in place but not the interface I/O code which would allow the user to make his design selections.) Further, we chose to allow the user to select multiple sequential faults, again within logical constraints. Thus, having executed a trial run with a particular fault, the user is then allowed to do one of three things:

1) Choose a fresh configuration (ISEE or IUE) and a fault for that otherwise normally operating HPS;

2) Choose a second fault for the same configuration, now operating in "degraded" mode;

3) Exit the system, finishing the demonstration run.

With the above-mentioned modifications, the system was implemented towards meeting the intent of the requirements document with regard to the fault handling module.

The fault handling capabilities identified above require considerable support software. The requirements document identifies an executive control module, an I/O or interface module, an internal model for representing the functional state of the HPS, and a telemetry generation module. These are treated in more detail in the design and implementation sections of this report and in the design document: the original intent for these modules has also been met.

It is appropriate to add that the telemetry generator was a requirement we had not expected at the outset of the contract. We had anticipated using historical tapes of actual, decommutated telemetry data from the ISEE (or IUE) spacecraft. It proved impractical to use such tapes. First, decommutated tapes were more difficult to access than tapes of raw data. Second, the historical tapes have relatively few fault symptoms on them, since there have been few significant failures on these spacecraft. Third, to use a set of canned data could lead to suspicions of a hardwired diagnostic system. Finally, if such data were used, it would be difficult to show the results of either the diagnostic actions (opening and closing valves and firing thrusters) or the workarounds. Unless the data were manipulated, the fault handler would have to perform actions that correlated with the data coming next on the tape. This would be quite unrealistic. Consequently, it was necessary to add the unanticipated task of developing a simulation program to generate telemetry. This turned out to be a considerable effort: the logic for the telemetry generator is quite complex--more complex, in fact, than that of the fault handler. This significantly impacted the development time.

IV. DESIGN PROCESS

This section will deal with the design process for the expert system. First, a system overview will be presented, which will discuss the overall operation of the system. Next, a brief description will be presented of the individual modules including a discussion of the rationale for the decompostion selected.

System Overview

The overall flow of control through the program is directed by the executive. After initialization of the program environment, the executive is invoked. It calls a part of the I/O module which queries the user for choice of propulsion system configuration, fault type, and fault location. Choice of configuration entails identifying a model of the propulsion system for internal use by the demonstration system. Once the user selects a configuration, either ISEE or IUE, the appropriate model is loaded from a file. When the fault is selected, the system's internal model of the HPS is modified accordingly for use by the telemetry generator.

The executive next calls the exposure module, which examines the fault selected in the context of the configuration. The exposure module then determines what state the propulsion system should be in to cause the fault to influence the operation of the spacecraft. This "exposure" state is one in which symptoms will appear in the telemetry stream as either abnormal or unexpected values. The exposure module identifies how the HPS should be "commanded" (using routine actions) to move into such a state and informs the user of what actions, if any, will be performed. It then returns a list of these actions to the executive.

The executive now calls upon the telemetry generator and passes to it the list of actions received from the exposure module. The telemetry generator modifies the internal model of the HPS to reflect the effects of these actions and calculates the appropriate telemetry values for the new state of the HPS. This telemetry is displayed to the user via the graphical depiction of the HPS and examined for unexpected values by a distributed monitoring system embedded within the telemetry generator module. Any abnormal or unexpected values are reported quantitatively with respect to the telemetry and expected values (e.g., 270 telemetry, 290 expected) and qualitatively with respect to the difference between the telemetry and expected values (e.g., "lower"). These values are incorporated into a list of symptoms which are then passed on to the fault handling module (FHM).

The FHM concludes as much as possible from the initial symptoms available and if it is unable to pinpoint the fault precisely it decides what actions will yield the necessary information to complete the isolation process. The FHM then describes what symptoms have been noted, any deductions which it has made, and what actions are being requested, if any. Next, the FHM returns to the executive module the list of actions to be performed. Thereafter, things proceed as they did after the exposure module returned its list of actions. That is, the executive passes the list of actions on to the telemetry generator, which modifies the internal model to reflect the actions being performed and calculates the new telemetry values. These values are screened to identify the abnormal and unexpected readings, and the results passed on to the FHM, and so on until the FHM has isolated the fault as precisely as possible. The FHM then informs the user of its conclusion, identifies the parts of the system which are unusable as a result of the fault and the workaround, and returns a list of actions which will effect the workaround. This list is passed to the executive and from there on to the telemetry generator for implementation.

When the workaround has been implemented and telemetry has normalized the user is given either two or three options, depending on the type of trial which has just completed:

a) the user can end this session;

b) the user can run a trial on either HPS configuration, beginning from a normal fault-free HPS; or

c) if the trial just completed was for the initial fault in an HPS, the user can select another fault in the same HPS. The HPS will begin in its current "degraded" state, retaining the original fault and its workaround, and then accept a second fault from the user.

In the first case, the demonstration program will terminate. In either of the latter two cases, the program will perform as described above.

Design of the modules

Every large program should have a module devoted to program control and data passing. In this case, the executive module performs this task. The executive calls procedures which then perform their function and return results to the executive. The executive takes these results, determines what data is needed by the next part of the program and calls the next procedure with the necessary data. This data may be whatever was returned from the previous procedure, or it may be a modified version of that data, or it may include data generated earlier. The executive is responsible for knowing the needs of the procedures which it calls and either itself modifying or having another procedure modify the available data so that it conforms to the specifications of the next procedure, in format as well as content. For example, the executive calls a procedure to put the list of symptoms in a certain order before the symptoms are made available to the FHM.

The exposure module is a module which is not absolutely essential but which is, nonetheless, very helpful for a demonstration system. The purpose of this module is to determine how the chosen fault can be exposed so that the system can begin at once to perform the isolation and subsequent workaround. While any fault chosen would eventually become noticeable if the HPS were asked to perform random routine actions, by requesting a specific set of routine actions the exposure module can enable immediate detection of one or more symptoms of the fault. As soon as symptoms are noticed the true focus of this project can occur, which is the expert system isolating the fault and then determining a workaround.

The expert system requires input from the user at several points in the demonstration and presents data to the user in many different ways. It was decided, therefore, to have an I/O module to deal specifically with this task of system-user interface. The I/O module contains several menus which are presented to the user to determine what HPS configuration, level of explanation, display of rules and choice of fault are desired. This module also contains the procedures used to display the HPS configuration on the screen and to insert the updated values as they are created by the telemetry generator.

Since the expert system will not be hooked up to actual satellite monitoring equipment, it was necessary to simulate the telemetry which would normally be downlinked from a satellite. This task is handled by the telemetry generator module. The telemetry generator uses structural information about the HPS, fault information, and knowledge of the previous values to decide what would be appropriate telemetry values for a given state of the HPS. This information is stored in an internal model (described below) and accessed by the telemetry generator so the new values can be computed. This module also has a telemetry monitor embedded within it. The telemetry monitor determines when the telemetry differs from expectations and composes a list of these values to be passed on as symptoms to the FHM. It could be argued that the procedures for telemetry monitoring should be contained in a separate module. If the expert system were using actual satellite telemetry data, then these monitoring procedures would be in their own module. Since the telemetry data must be simulated, however, and since the telemetry monitor uses many of the same procedures as the telemetry generator to access values from the internal model, these two functions were combined into a single module.

The internal model of the HPS contains information about the structure of the current HPS model, the current fault selected by the user (and the previous fault, if any), and the various values for each monitored attribute of the system. The structural information deals with the number of each type of component, and which components are above and below a given component. The fault information reveals whether a given component has a fault associated with it, and specifies the exact nature of the fault. Each monitored attribute in the internal model has three values associated with it. These values are:

- Actual value - the value used by the telemetry generator to define the true value of the attribute being measured.

- Telemetry value - the value reported by a sensor which is supposed to measure the attribute. This will be the same as the actual value unless the sensor is faulty.

- Expected value - the value which should be reported by a sensor if every part of the system which affects the attribute and the sensor itself are functioning normally. If a fault exists which affects either the attribute being measured or the measuring sensor, then the telemetry value will differ from the expected value.

The fault information and the actual values are accessible only by the telemetry generator, which needs this information to accurately determine the effects of a fault when generating the telemetry values. The other information can be accessed by all modules, since such information would be kept in an internal model even if the system didn't need an internal model for simulating the data from a satellite. Although maintaining two distinct models would emphasize what information is both needed for and available to only the telemetry generator, it is easier to have two distinct conceptual models which are implemented as a single model with the understanding that most modules do not access certain data in this model.

The fault handling module (FHM) contains the code to perform the actual isolation of the fault and determine the appropriate fault workaround. This module consists of four major components:

- The knowledge base - this consists of rules which encompass the "expert" knowledge about the effects and symptoms of faults, common fault isolation techniques, and appropriate fault workarounds.

- Isolation procedures - these procedures are called by rules in the knowledge base to determine what specific actions are needed to assist the isolation process and to compose this list of actions which is returned from the FHM.

- Workaround procedures - these procedures are used to help determine what actions are needed to enable the HPS to recover from the fault as fully as possible. There is also code here which cleans up the knowledge base after the fault is isolated. This code makes certain that only essential knowledge is retained.

- Explanation procedures - these procedures handle the explanation presented to the user. They are used by both of the above components to describe what has been deduced, what actions are being requested and why, and the final results of the isolation and workaround.

V. IMPLEMENTATION EXPERIENCE

This section describes the tools used in implementing the expert system and the process of moving from the design to the implementation. The latter includes a description of some of the changes from the original design made during the implementation phase of the project and the rationale for these changes.

The implementation tools

The entire expert system, except for the knowledge base, was written in the programming language Franz Lisp. At this time, Lisp is the most prevalent programming language for AI applications and Franz Lisp was chosen because it is the Lisp dialect residing on the GSFC VAX 11/780, which was the target machine for the demonstration phase of this project.

The knowledge base was written in MRS, which stands for Modifiable Representation System. MRS was used because it has rule base capabilities which facilitate the storing and accessing of information in a partitioned format. This ability is useful for several reasons. A rule format is good for storing expert knowledge because such knowledge can generally be easily expressed in statements of the form IF [one or more conditions are true] THEN [do one or more things]. Rules are also useful because they are easily modified or added to a knowledge base without requiring modification of the rules already there.

MRS also allows pattern-matching of data to rules, which allows a rule to be more general than the simple IF clause of typical programming languages. Pattern-matching enables a rule to specify certain slots in a data item as needing to match exactly with part of the rule and other slots to be any value. Thus, a rule which is intended to work whenever any latch valve reports any symptom can look for a data item which contains the component name "lv" (for latch valve) and not worry about the component number or position of the valve. In the HPS for ISEE, there are four latch valves, so having one rule to check for any latch valve symptom as opposed to one rule for (lv 1 open ...), a second rule for (lv 1 closed ...), a third rule for (lv 2 open ...), etc. allows one rule to do the work of eight.

Since MRS allows the rules and data to be partitioned, using it is also more efficient than using regular Franz Lisp code for the knowledge base. The rules are grouped together according to class of fault (e.g., latch valve, catalyst bed, engine valve, etc.) and stored in the knowledge base according to these groups. Once initial isolation is done and the expert system knows to which group the fault belongs, it checks only the rules in that group and ignores rules in the other groups. By diminishing the number of rules checked for applicability, the run-time speed of the program is increased.

The implementation process

Once the design of the modules was completed and the system overview describing the flow of control between modules was done, work on the actual implementation of the system began. This consisted of assigning the various modules to specific programmers, with constant intercommunication so that there wouldn't be confusion about the format or content of data passed between modules. During this process, it became evident in some cases that a task initially assigned to one module could be implemented more efficiently within a different module. On these occasions the programmers involved would agree to transfer responsibility for the task in question and any code already written by the original programmer would also be transferred.

Changes made during implementation

As mentioned earlier, there were occasions during the implementation of the expert system when changes from the original design were made for various reasons. Below are examples of what was changed during the implementation of the system and a justification of why each change was deemed necessary or worthwhile.

Change #1: Modifying the internal model

Modifying the internal model of the HPS was originally designed into the executive module. It was decided to have the telemetry generator perform this task, rather than the executive, for two reasons.

First, the telemetry generator already needs access to the internal model to produce the telemetry values. To do this, the telemetry generator has to have complete access to the internal model, including the actual values as well as the telemetry values (sensor readings). Since an expert system would normally have access only to the telemetry values and not to actual values, it was decided to minimize the number of modules which did access the actual values. In this system, actual values are accessed only when modifying the internal model and when calculating the telemetry values. Therefore, it made sense to have the telemetry generator perform any necessary modifications to the internal model and thus be the only module that needed to see the actual values. Since the telemetry generator is already called whenever the model is modified so it can calculate the new telemetry values, having it perform this additional task required no change in the overall flow of control.

A second reason for moving this task out of the executive is that such a task is realistically beyond the scope of an executive's responsibilities. The executive is intended to be a high-level controlling module which passes data and calls procedures while delegating lower-level tasks (such as the modifying of the internal model) to the subordinate modules.

Change #2: Describing actions that the HPS performs

Describing the actions being performed by the HPS was also originally designed as a task for the executive module. This has been changed so that it is now done by the module responsible for requesting the action. There are two different modules which can request actions to be performed in the HPS and are, therefore, responsible for describing these actions to the user.

The first of these is the exposure module. After the user specifies the desired fault, the exposure module passes to the executive the list of actions chosen to bring symptoms of the fault to light. Before passing this action list, however, the exposure module informs the user of what actions will be done. If no actions are being performed to expose the fault, as is sometimes the case, the user is informed of this fact.

The other module that is able to request actions is the fault handling module (FHM). This module can request various actions when it needs more data to isolate the fault. The FHM also composes the list of actions necessary for the fault workaround. Like the exposure module, the FHM informs the user of the actions that will be performed before passing the action list to the executive module.

Changing where the description of actions takes place was done for two reasons. First, the module creating the list of actions will know exactly what is being done and why and, therefore, can readily inform the user of this. If the description of actions were to be done by the executive or another module other than the one requesting the actions, one of two things would be necessary. Either the description would have to be composed by the requesting module and passed along with the action list, or the module receiving the list of actions would have to examine the list to find out what was being requested, and then try to figure out why those actions were being requested. Neither of these methods are as straightforward as having the module that requests the actions perform the description of those actions. Also, the latter method could entail a significant amount of computation while still yielding an incomplete or inaccurate description.

The second reason for not having the executive responsible for the description of actions was mentioned to some extent previously. The executive's job is to receive and pass data from those procedures that actually perform the computations. Therefore, requiring the executive to know what that data is should be unnecessary. The executive should be able to pass data among the modules without worrying about the specifics of what it is passing. By moving responsibility for the description of actions being performed to the requesting modules, the executive doesn't have to examine the list of actions before passing them on to the telemetry generator.

Change #3: Allowing two sequential faults

The original proposal called for the system to handle a single fault in an HPS by isolating the fault, working around it, and then going to a new HPS. It was subsequently decided to enhance the system by adding a multiple-fault handling capability. The handling of two simultaneous faults was dismissed for two reasons.

First, the combinatorial difficulties of allowing the faults in the HPS to be simultaneous would greatly complicate the expert system. Two faults could have offsetting symptoms, different combinations of faults could have the same combinations of symptoms, and there may be no way of isolating certain combinations of faults given the amount of information available.

Another consideration is that faults in a satellite's HPS are very rare. This is due to the emphasis on quality construction when something is to be launched into space, far away from the nearest serviceman. The chance of two things going wrong simultaneously was deemed to be fairly insignificant and the additional effort of dealing with simultaneous faults was not justifiable. Handling two sequential faults, however, was judged to be feasible. Knowledge of the initial isolated fault enabled the system to reason about how that fault might influence the symptoms of other faults, and also allowed it to determine when it might need to use a different process for isolation or workaround of a sequential fault. Therefore, the ability to handle two sequential faults, while requiring more work than that necessary for single faults, was not excessive. Also, the possibility that a second fault might occur sometime during the entire lifespan of a satellite is high enough to warrant consideration, as it is greater than the odds of two simultaneous faults. Therefore, the arguments against two simultaneous faults do not fully apply to the case of two sequential faults.

It was decided to deal with only two sequential faults, rather than some greater number, for several reasons. First, since the HPS is assumed to be fairly reliable, the probability of increasingly greater numbers of faults diminishes rapidly. Conversely, the complexity required in the expert system for handling greater numbers of faults increases rapidly. There is, therefore, a greatly decreasing return on the effort invested in handling more than two sequential faults. Also, the presence of three or more faults in an HPS generally renders the HPS unusable regardless of the possible workarounds, so attempting to handle more than two faults in the same HPS is all too often an empty gesture.

VI. AUTOMATION

An evaluation of the Multiple Satellite Operations Control Center (MSOCC) led to the conclusion that automation of several aspects of this center is both feasible and highly desirable. If this automation is implemented in conjunction with a network of computer workstations, to be used by the center's analysts, the benefits will increase. These benefits include increases in efficiency and reliability, as well as the solution of several existing problems which will be detailed below.

One problem is that certain data and procedures may be stored in only one location within the MSOCC, so they are less accessible to analysts in other parts of the center. This locality of information, especially with procedures for dealing with various situations, could result in analysts remote from the procedure storage location trying to remember or guess at what to do in situations where a particular procedure should be followed.

The standard way to attack this problem is to maintain copies of needed data, procedures, etc., throughout the MSOCC, thereby assuring a copy will be convenient for any analyst requiring this information. This solution, however, leads to a different problem. Having multiple copies of information scattered about makes it difficult to organize and update the information. It does little good to have several locations where information is available, if the information there may be incomplete, inconsistent or outdated.

A network of computer workstations distributed throughout MSOCC is recommended because such a network would avoid both of the above problems. Having on-line files which contain the satellite data and the policies and procedures for dealing with common situations will result in easy access to this information by any analyst who requires it. (This does not mean that access to information can no longer be controlled. It is straightforward to restrict access to certain files according to user.) Moreover, only one primary on-line copy of each file of information is needed. This simplifies the organization and updating of the information.

Other problems can result from having information divided among several different storage mediums. Information is currently found in filing cabinets, on computer tapes, in engineering drawings, or in manuals.

One problem with this is that it requires any analyst who uses this data to be familiar with all of the mediums used to store it. This includes familiarity with the filing system, loading tapes, reading drawings, etc.

Another problem is that a variety of mediums tends to result in a variety of storage formats. There is a tendency for each creator or maintainer of information to use whatever format he or she prefers, and thus it is often difficult for another analyst to rapidly assimilate information from different sources. The use of different storage mediums necessitates some difference in storage formats and, therefore, it is difficult to discourage analysts from varying the format even further to suit their preferences.

These problems are also addressed by having a network of workstations to contain information. There would be only one storage medium used on a regular basis—computer tapes would be used for backup purposes only and hard copies (printouts) of the information are maintained in case of hardware problems with the computer network. Therefore, analysts need only be familiar with the use of the workstations. Also, having single copies of information on the network facilitates enforcing a single storage format, so processing the information would be faster and easier than before.

Automation of repetitive, simple tasks can enhance efficiency in two ways. First, computers typically can perform simple tasks such as range checking much faster than a human. Second, a computer cannot become distracted while performing a task, so it can be relied upon to not overlook incoming data. It is also dependable when it comes to remembering to log data and actions as required.

There are several different types of tasks which would be suitable for automation, especially with the use of a network of workstations. These tasks include: telemetry monitoring, decision making in both known and novel situations, scheduling of ground coverage and experiments, and management of experiments. A more detailed discussion of possible automation of these tasks can be found in Appendix D of this report.

Other uses for automation would include automatic report generators and data formatters. These would allow rapid creation of transcripts for processing by the analysts, facilitate the updating of databases, and allow easy generation of procedure manuals based on actions taken in response to new situations. Such programs could use the results of the other automated components of the system.

Standard automation such as telemetry monitoring would also facilitate the introduction of more sophisticated programs, such as expert systems to perform diagnosis or planning. This is because the output of the standard automation is often in a format which is readily usable by an expert system. Such an interface is helpful for two reasons.

First, a sophisticated user interface for input to the expert system is not required. It is relatively simple to have a procedure perform any needed modifications to the output of a telemetry monitor, for example, to change the data into the input format used by an expert system that performs fault analysis. It would would generally be more difficult to have an analyst who was monitoring the data create the inputs to the expert system, since the expert system may then have to accommodate typing errors, transposed parameters, etc.

Another advantage to having an automated system provide inputs into an expert system is that it can pass the information much faster than an analyst could type it. For programs that need to operate in a real-time environment, this is clearly a point in favor of automation. If the expert system needs to interact with a monitoring agent (i.e., check for the presence of certain telemetry, request changes in the scope of the monitoring), removing the comparatively slow analyst from the chain increases the run-time speed of the expert system even more dramatically.

VII. RESULTS

Our experience with constructing the demonstration fault handling system brought a spectrum of results from an understanding of satellite operations control center activity to an illustration of the use of artificial intelligence languages for simulation. These will be discussed in turn.

a) As indicated, the detailed interaction with members of the ISEE and IUE control centers evolved an in-depth understanding of many control center activities. This understanding allowed us to identify different automation approaches for different aspects of each activity. We explored several in detail, including procedure generation, command generation, and scheduling, before centering on fault handling.

b) A recommendation emerged for an in-depth analysis of the automation of each and all aspects of a control center. Computer Technology Associates has generated an abstract study of this nature, and we propose, as well, an analysis in the concrete terms of a particular control center. Together with this recommendation, we have presented a high-level description of one possible approach to that automation (see Section VI). This is, in effect, a global control mechanism or strategy that would coordinate other automated elements.

c) We have developed a demonstration software system to perform fault handling on the hydrazine propulsion system of either the ISEE or IUE spacecraft. This system runs in Franz Lisp and is presently hosted on the VAX 11/780 and 11/750 series machines. With appropriate memory resources and terminal configurations (a VT100 compatible terminal), it could be hosted on other machines, including the "supermicros," running Franz Lisp.

d) The demonstration system exemplifies how "production rules" can be used as a programming language. The rules are written in the MRS tool which runs under Franz Lisp. During a trial run of the system, the user can elect to see the rules printed out, either in their general form with all variables unbound or as they have matched to specific data elements in working memory. (An annotated example of rule printing is included in Appendix C.)

In the first of these cases, the user can obtain an appreciation for the generality and expressive power of a rule--how it can be used to cover a number of situations. In the second case, the user sees the rules in their "instantiated" form. In this form, with all the actual data values inserted into the appropriate places in the rules, the user can understand how that rule is functioning in a specific situation. This aids in following the internal logical flow of the fault handling process.

e) The demonstration system also illustrates the use of artificial intelligence languages for simulation. The unanticipated requirement of building a simulation to generate telemetry afforded us this opportunity. While a variety of AI tools could have been used for this, we opted to build the simulation directly in Franz Lisp. Had we been working on a Lisp Machine, we might have chosen to implement the simulation in the object-based language Flavors or, if we had been working in Interlisp, in ROSS, an experimental AI language designed to build simulations. Unfortunately, the user can witness little of the inner workings of the telemetry generation: it was designed so that only the telemetry values that it outputs are displayed to the user.

f) The system presents the possibility of using English to describe the system analytic process. The user has a choice of what level of description to see. It can be turned off almost entirely, or a full description can be presented. The system can thus depict for the user how it is making various deductions and conclusions and describe the actions it is requesting.

g) Documentation of how this system has been developed has been generated in the various interim reports and deliverables and in this report. This includes:

- o choice of the application domain;
- o the set of faults the system would handle (this is the high-level requirements specification for the rule-based fault handling module);
- o requirements for the support system;

- o preliminary design;
- o modifications to the initial design conception; and
- o implementation.

h) Developing the system provided an expanded understanding of development time considerations for AI-type systems. Additions to and changes in performance specifications complicated this issue.

i) The experience solidified the understanding of the importance of development environments for AI-type systems. The facilities provided by a Lisp Machine running, say, Zetalisp, greatly increase the ease of understanding how a piece of code is functioning and therefore how to fix its bugs. The facilities resident in Franz Lisp are of almost no assistance and only in-house debugging tools aided in this process.

Further, the graphics capabilities inherent in a Lisp Machine and other personal work stations are highly sophisticated, simple to use, and run with minor overhead. Hosting a full-blown graphics package on a VAX for use with a VT100 series terminal adds a significant processing load. Using intricate programming tricks, we were able to provide a pleasant graphics display with almost no overhead. We were able to do this only because the graphics never changed: had the option of letting the user design his own configuration been retained, it would not have been feasible to provide a graphic display for it without using a special graphics package with its significant associated overhead.

j) We chose MRS, the Modifiable Representation System, as our inferencing tool because it fulfilled our objective better than other reasoning tools available on the VAX. MRS is not a traditional production system such as OPS5, EMYCIN or HAPS. It has a variety of powerful representation and reasoning capabilities, including the forward chaining ability which we used to drive our rules. Unfortunately, MRS is not designed to do forward chaining in a way that is convenient to the system builder (a similar statement can, regrettably, be made about other reasoning tools as well).

Thus there are, for example, cases where extra rules or extra clauses in rules must be inserted to perform bookkeeping and cleanup actions. Under optimal conditions, a tool of this nature would perform such activities in a way the user would not even need to know about. A final result, then, of this experience relates to the appropriateness of the development tools independent of the development environment. The conclusory results are detailed in the next section.

VIII. CONCLUSION

In the course of developing the demonstration fault handling system for this contract, we identified a number of considerations about control center automation in general. Developing this particular system entailed a detailed examination of control center activities. It is clear from this that most of these activities could be automated. Ideally, this automation should be designed in as an integral aspect of the center operations during conception and development. Retrofitting extensive automation to an existing configuration can be expensive and inefficient. However, adding automation need not be an intensive, disruptive process. Automation can be added incrementally to an existing center with minimal inconvenience and commitment. The benefits at each stage of automation would then be evaluated to determine if further automation is justified.

More important, perhaps, is the question of what techniques should be used to automate the different activities. Approaches run the gamut from hardware components (e.g., sensors that control heater activity thermostatically) to conventional software (using, e.g., decision trees in C or FORTRAN) to artificial intelligence (using perhaps a production system or an automated planning tool). In deciding what approach to take, it is useful to consider that automation is important for (a) labor intensive activities and (b) activities requiring considerable, detailed knowledge or expertise.

When an activity reaches a certain level of either repetition or complexity, automation is appropriate. The level of automation should be matched to the type and priority of the task. Most control center activities fall under (a) above and are generally automatable by conventional technology. Those in the second category may be potential candidates for automation using artificial intelligence. There are a limited number of these in the existent control centers within the MSOCC at GSFC. These include global coordination and executive functions and situations requiring causal or deep reasoning using an understanding of, perhaps, physics and astrodynamics. In these situations and in the scheduling activities, however, there are political concerns which would need to be addressed.

In any case, the tradeoff between the cost of automation and the benefit--viewing each from both a short-term and long-term perspective--is very important. The life expectancy of many spacecraft may not be long enough to warrant the extensive automation of activities specific to a particular spacecraft. Some of the processing tasks such as scheduling and fault handling, however, are more generic, and systems to automate these could be used for several different spacecraft, both simultaneously and across time. It is important as well to consider the potential for the ongoing evolution of an automated system. It could simply be extended to do more. Alternately, it could be a somewhat general system which, with a small effort, could be adapted to perform the same activity in different situations. These and other forms of evolving a system would create a cost factor independent from any ongoing maintenance. These more generic and/or evolvable activities then may be the most appropriate place to focus an automation effort. Such a focus would produce the greatest long-term benefit: a multi-purpose and "reusable" automated system. Unfortunately, this leaves a dilemma: while many of the more labor-intensive tasks (such as telemetry monitoring) are generic, the activities requiring the greatest experience and expertise tend to be highly spacecraft specific--for example, a causal reasoning system to troubleshoot a particular subsystem or, even more so, a particular experiment.

Thus, the decision to automate is a complex consideration of several factors which require close examination and careful balancing. As construction costs for such systems come down, the cost benefit ratio will improve, especially in the long term. Now, longevity of use is perhaps the major consideration: over what time frame will the fixed construction costs plus any ongoing maintenance costs be spread? Is that cost greater than the total cost would be for human labor over the same time period if the automation were not done?

Additional risks must be considered such as the loss of a satellite for which spacecraft-specific automation is done. (This again will be influenced by the generality and adaptability of the automated system.) Without such examination, inappropriate automation decisions, and hence expenditures, might be made.

APPENDIX A

Glossary of Terms

Glossary

AI	Artificial Intelligence.
Assert	to place data in working memory. This action can result in rules "firing" (see Fire).
Assertion	a data item which has been placed in working memory.
Catbed	Catalyst bed, see the HPS component descriptions on page A - 3.
CB	Catalyst Bed, see the HPS component descriptions on page A - 3.
EV	Engine Valve, see the HPS component descriptions on page A - 3.
Expert system	a computer program designed to "intelligently" perform some task by utilizing the knowledge and practices of one or more experts in the task domain.
FHM	Fault Handling Module.
Fire	term used to describe the process whereby the preconditions of a rule are satisfied, causing the consequents of the rule to be executed.
GSFC	Goddard Space Flight Center.
HPS	Hydrazine Propusion Subsystem.
ISEE	International Sun-Earth Explorer, a satellite controlled by GSFC.
IUE	International Ultraviolet Explorer, a satellite controlled by GSFC.
LLV	Lower Latch Valve, refers to the lower set of latch valves in IUE. See the latch valve component description on page A - 3.
LV	Latch Valve, see the HPS component descriptions on page A - 3.

MOSES - Mission Operations Support Expert System, the demonstration fault handling system built under this contract.

MRS - Modifiable Representation System, a multi-purpose tool for modeling and representing knowledge. Our system uses the rule-based features of MRS to incorporate knowledge into a collection of rules.

MSOCC - Multiple Satellite Operations Control Center.

Rule - a format for encoding knowledge, usually consisting of a set of preconditions followed by a set of actions to be performed if the preconditions are satisfied.

TG - tank group, see the HPS component descriptions on page A - 3.

THR - thruster, see the HPS component descriptions on page A - 3.

ULV - upper latch valve, refers to the upper set of latch valves in IUE. See the latch valve component description on page A - 3.

Workaround - a set of actions designed to negate or minimize the effects of a fault on the future performance of the system.

Working memory - a specific location in memory for data to be placed so that it can be accessed by the rules as needed.

The components which compose an HPS are as follows:

- o Tank groups - groups of tanks which store liquid hydrazine fuel on the satellite. (See Figure A-1)
- o Fuel lines - lines which carry the liquid hydrazine throughout the propulsion system (see Figure A-2).
- o Fuel line heaters - heaters found along most of the the fuel lines, they prevent the liquid hydrazine from freezing inside the lines.
- o Latch valves - valves which are placed in strategic locations within an HPS. They are normally open, but can be closed to isolate sections of fuel line during the fault isolation and workaround processes.
- o Engine valves - valves which open briefly, causing fuel to flow over a catalyst bed, and then automatically close. They are used to fire the thrusters.
- o Catalyst beds - these are very hot beds over which the fuel is passed, causing the liquid hydrazine to change to gas and produce a thrust.
- o Thrusters - nozzles through which the ignited hydrazine gas is forced, giving the satellite the intended thrust (see Figure A-3).

KEY TO DIAGRAMS

|‾_|	-	Fuel tank
|||	-	Valve, open (usual state of latch valves)
|-|	-	Valve, closed (usual state of engine valves)
|_| / \	-	Catalyst bed and thruster

An example of tank groups is shown below:

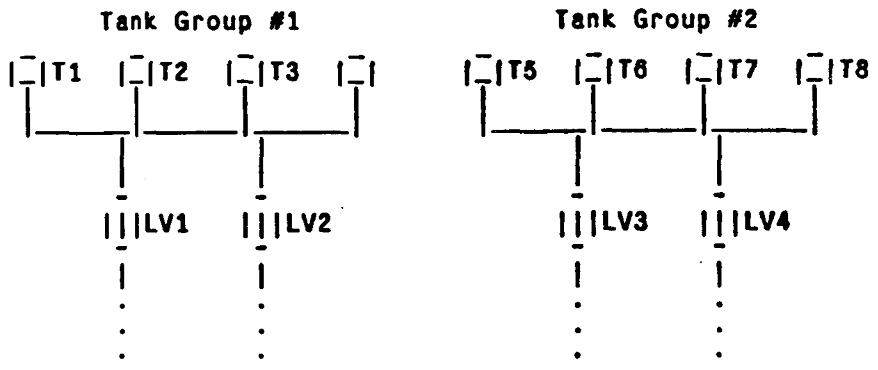


Figure A-1 (Part of ISEE hydrazine propulsion system)

Even though there are eight separate tanks, there are only two functional tank groups. Each group has four tanks which are connected via the same fuel line, and have the same group of components directly below them. Tank Group #1 consists of (T1 T2 T3 T4) with (LV1 LV2) below and Tank Group #2 consists of (T5 T6 T7 T8) with (LV3 LV4) below.

An example of fuel line is shown below:

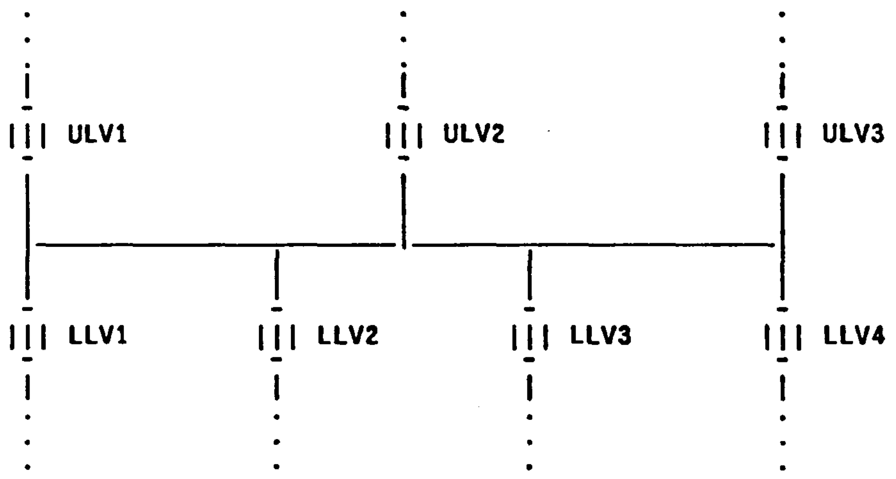


Figure A-2 (Part of IUE hydrazine propulsion system)

A fuel line is specified by the components at either end of it. It is possible for a section of line to have several components at either end, as seen above.

An example of the path to the thrusters is shown below:

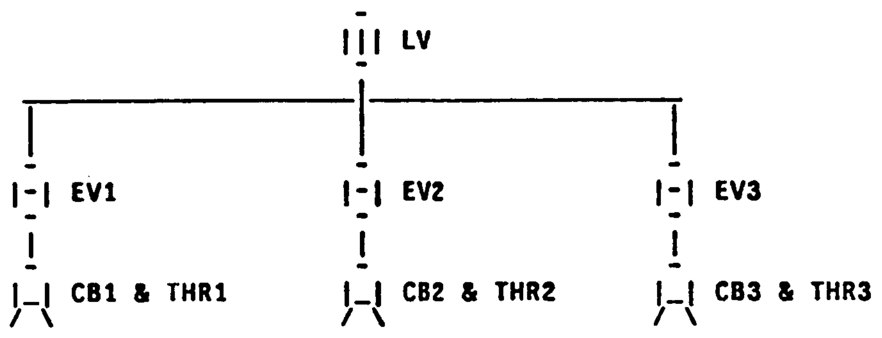


Figure A-3   (Part of IUE hydrazine propulsion system)

When an engine valve opens, fuel flows to the catalyst bed, where it becomes a gas, is ignited and forced out through the thruster nozzle.

APPENDIX B

Assumptions

Assumptions

The temperature outside the spacecraft is assumed to be a constant. This avoids requiring the telemetry generation module to know when the spacecraft is in the sun and when it is in the shade. This in turn permits it to assume constant heat dissipation when the fuel line or catalyst bed heaters are off, until the temperature stabilizes.

Two other assumptions concern the rate of fuel loss. It is assumed that all fuel leaks are of a uniform size/rate. This allows the expert system to assume a single constant rate of decrease in pressure when a leak is present. Similarly, each firing of a thruster is assumed to use a fixed amount of fuel from each tank group, resulting in a uniform decrease in pressure.

There are also assumptions about the location of sensors within the HPS. The sensors are located such that all faults can be identified, yet faults cannot be pinpointed immediately in cases such as line failures. This allows demonstration of the expert system "searching" sections of fuel line before isolating a line leak or heater failure precisely. Below is a table containing the locations of various sensors.

TYPE OF SENSOR	LOCATIONS WITHIN THE HPS
Pressure	One within each tank group (above the latch valves).
Temperature - fuel line	One immediately above each engine valve.
Temperature - catbed	One within each catalyst bed.
Open/close position	One at each latch valve and at each engine valve.

There is no redundancy in the sensor for a given component or area. Since the expert system can identify and isolate sensor failures, redundancy in sensors was not deemed necessary.

APPENDIX C

Example of a Fault Being Isolated

This is an example of the system isolating a pressure sensor fault. The fault which has been selected is the sensor at tg1 (tank group 1) reading lower than the actual pressure.

(Although the display of the HPS is not shown, the system's explanation, the symptoms asserted to the FHM, the actions requested from the FHM, and the rules being fired are all shown. Some comments to assist the reader in understanding the process of rules firing and the source of explanation will be enclosed in angle brackets, "<" and ">")

No action will be performed in exposing this fault.
< This description, and the action list, come from the exposure module >
< List of actions passed to telemetry generator: none >

The assertion-list is: ((tg 1 lower 236 300))
< This list is composed by the distributed telemetry monitor embedded within the telemetry generator and then passed to the executive. The assertion list is composed of a set of sublists (in this case only one), each representing a telemetry point whose sensor reading differs from its expected value. Here, the sensor at tank group one reports a lower pressure than expected. The sensor reading is 236 when it was expected to be 300. The executive asserts each item on the assertion list into the working memory of the FHM. >

The assertion being asserted now: (tg 1 lower 236 300)

< Below is the first rule that fired. >

```
(if (and (tg |$n| |$q| |$t| |$e|)
         (unknown (ignore-symptom tg |$n| |$q|)))
    (runnable (assert-and (and (print-rule p331 |$n| "$n" |$q| "$q"
                                                |$t| "$t" |$e| "$e" x)
                               (tg-symptom |$n| |$q| |$t| |$e|)))))
```

< Below is the binding list of each variable in the rule and its value. (Variables are encased by vertical bars and contain a dollar sign "$"). By looking at the "if" portion of the rule and the preceding assertion, the reader can see how the values for the variables were found. In this rule, the system looked for any assertion that had five slots, and the first slot had to be "tg". When the above assertion was asserted (put into working memory), the "tg" matched to the above rule, and the assertion had five slots. When the rule was "instantiated" (the "if" part matching to the assertion), the variable |$n| was bound to the value 1, since they were in the second slots of the rule and assertion, respectively. The |$q| in the third slot of the rule was bound to "lower" in the third slot of the assertion, the |$t| was similarly bound to 236, and |$e| to 300. The second part of the "if" clause means that if there is an assertion present whose first slot is "ignore-symptom", second slot is "tg", and last two slots are whatever |$n| and |$q| were bound to (in this case "1" and "lower"), then the above rule should not fire. Since there was no such assertion present (such an assertion was "unknown") the rule was able to fire, creating the binding list shown below. >

```
(|$n| 1 |$q| lower |$t| 236 |$e| 300)
```

< Below is the next rule to fire. >

```
(if (and (tg-symptom |$n| |$q| |$t| |$e|)
         (unknown (fault |$dum1| |$dum2| |$dum3| |$dum4|)))
    (runnable (assert-and (and (print-rule p332 |$n| "$n" |$q| "$q"
                                                |$t| "$t" |$e| "$e" x)
                               (initial-describe tg |$n| |$t| |$e|)
                               (activate-theory pressure-symptom)
                               (fault pressure |$n| |$q| dummy)))))
```

< Below is the binding list of each variable in the rule and its value. By looking at the "if" portion of the above rule and the latter part of the preceding rule, the reader can see how the values for the variables were found. The preceding rule caused "(tg-symptom 1 lower 236 300)" to be asserted into working memory (the last four slots of the list containing the values of the variables), and this assertion matched the "if" part of the above rule, with the variables in the above rule receiving the values of their respective slots in the assertion. >

```
(|$n| 1 |$q| lower |$t| 236 |$e| 300)
```

```
tg1 has reported a fault symptom.
The sensor indicates a pressure of 236 when 300 was expected.
```

< Printed as a result of "(initial-describe ...)" in the above rule. >

< Below is the next rule to fire. >

```
(if (and (fault pressure |$n| |$q| dummy)
         (continue-fc)
         (unknown (cb |$m| higher |$t| |$e|))
         (unknown (thrust |$norm|)))
    (runnable (assert-and (and (print-rule p334 |$n| "$n" |$q| "$q" x x x x x)
                               (remove-one continue-fc)
                               (describe-why-fire)
                               (fire-random-thruster)))))
```

< Below is the binding list of each variable in the rule and its value. By looking at the "if" portion of the above rule and the latter part of the preceding rule, the reader can see how the values for the variables were found. >

```
(|$n| 1 |$q| lower)
```

```
A thruster will be fired to determine whether the fault affects the
attitude adjustment. This is done to help discriminate between sensor and
physical faults.
```

< Printed as a result of "(describe-why-fire ...)" in the above rule >

```
thr1 is now being fired.
```

< Printed as a result of "(fire-random-thruster ...)" in the above rule >

< List of actions passed to telemetry generator: ((ev1 open) (ev1 close)) >

The assertion-list is: ((thr 1 normal 100 100) (tg 1 lower 228 290))

The assertion being asserted now: (thr 1 normal 100 100)

< Below is the next rule to fire. >

```
(if (thr |$n| normal |$t| |$e|)
    (runnable (assert-and (and (print-rule p306 |$n| "$n" |$t| "$t"
                                                |$e| "$e" x x x)
                               (describe-normal-thrust normal |$t|)
                               (thrust normal)))))
```

< Below is the binding list of each variable in the rule and its value.

By looking at the "if" portion of the rule and the above assertion,

the reader can see how the values for the variables were found. >

```
(|$n| 1 |$t| 100 |$e| 100)
```

There was normal thrust, as expected.

< Printed as a result of "(describe-normal-thrust ...)" in the above rule >

The assertion-list is: ((tg 1 lower 228 290))

The assertion being asserted now: (tg 1 lower 228 290)

< Below is the next rule that fired. >

```
(if (and (tg |$n| |$q| |$t| |$e|)
         (unknown (ignore-symptom tg |$n| |$q|)))
    (runnable (assert-and (and (print-rule p331 |$n| "$n" |$q| "$q"
                                                |$t| "$t" |$e| "$e" x)
                               (tg-symptom |$n| |$q| |$t| |$e|)))))
```

< Below is the binding list of each variable in the rule and its value.

By looking at the "if" portion of the rule and the above assertion,

the reader can see how the values for the variables were found. >

```
(|$n| 1 |$q| lower |$t| 228 |$e| 290)
```

< Below is the next rule that fired. >

```
(if (and (fault pressure |$n| |$q| dummy)
         (continue-fc)
         (unknown (cb |$m| higher |$t| |$e|))
         (thrust normal))
    (runnable (assert-and (and (print-rule p336 |$n| "$n" |$q| "$q"
                                            x x x x x)
                               (sensor-deduction fuel-pressure
                                                 normal
                                                 pressure sensor)
                               (pressure-sensor-fault |$n| |$q|)))))
```

< Below is the binding list of each variable in the rule and its value. By looking at the "if" portion of the above rule and the latter part of two of the preceding rules, the reader can see how the values for the variables were found. >

```
(|$n| 1 |$q| lower)
```

```
Since there was a fault involving the fuel-pressure and firing the thruster
resulted in normal attitude adjustment, a pressure sensor fault is deduced.
```

< Printed as a result of "(sensor-deduction ...)" in the above rule >

< Below is the next rule to fire. >

```
(if (pressure-sensor-fault |$n| |$q|)
    (runnable (assert-and (and (print-rule p342 |$n| "$n" |$q| "$q"
                                            x x x x x)
                               (add-another-ignore-symptom tg |$n| |$q|)
                               (deactivate-theory pressure-symptom)
                               (wrap-up tg |$n| sensor-failure |$q|)))))
```

< Below is the binding list of each variable in the rule and its value. By looking at the "if" portion of the above rule and the latter part of the preceding rule, the reader can see how the values for the variables were found. >

```
(|$n| 1 |$q| lower)
```

< Below is the next rule that fired. >

```
(if (wrap-up |$c| |$n| |$f| |$v|)
    (runnable (assert-and (and (print-rule p303 |$c| "$c" |$n| "$n"
                                                |$f| "$f" |$v| "$v" x)
                               (fault-describe |$c| |$n| |$f| |$v|)
                               (workaround |$c| |$n| |$f| |$v|)
                               (wait-cycles variable)
                               (workaround-done)))))
```

< Below is the binding list of each variable in the rule and its value. By looking at the "if" portion of the above rule and the latter part of the preceding rule, the reader can see how the values for the variables were found. >

```
(|$c| tg |$n| 1 |$f| sensor-failure |$v| lower)
```

```
The fault isolation process has completed successfully.
The pressure sensor for tg1 reads lower than the actual pressure.
```

< Printed as a result of "(fault-describe ...)" in the above rule >

```
This fault requires that the system note that the sensor is faulty. No other
actions are required, and the entire HPS remains usable.
```

< Printed as a result of "(workaround ...)" in the above rule >

```
The fault workaround has been successful. The HPS should be able to resume
operation.
```

< Printed as a result of "(workaround-done ...)" in the above rule >

APPENDIX D

Automation Choices for the ISEE Control Center

AUTOMATION OF THE ISEE CONTROL SYSTEM

Activities in the ISEE control center appear to decompose into six categories: (1) telemetry monitoring (for both experiments and spacecraft operations); (2) deciding what action to take in response to known problems; (3) deciding what action to take for novel problems; (4) ground coverage scheduling, (5) experiment scheduling; and (6) experiment management. The automation of these activities will be discussed in turn.

I. TELEMETRY MONITORING

Telemetry is currently monitored by a shift analyst who stares for extended periods at a computer display attempting to catch telemetry readings that are out of limits or trending toward a limit condition. When he identifies such a situation he notifies the project analyst who makes all decisions about how to respond. The task of monitoring is tedious and mechanical. The shift analyst knows what the telemetry value ranges should be and what their limits are. He scans the actual values checking to see if they fall in the expected ranges. Trend identification is only slightly more difficult, requiring that the shift analyst remember roughly what previous values were. He can verify a trend if he thinks one exists by quickly checking the previous values.

Both of these shift analyst tasks could be readily automated using conventional software. Range checking would involve simply comparing each value to its expected range--a Fortran DO loop with a subroutine branch for out of limit conditions would suffice amply. For trend analysis, each telemetry value subject to numerical variability could have an evaluation function looking for the kind of trend that value might exhibit. The current value would be correlated with whatever number of previous values are required to establish the numerical trend. Non-numerical trends would be more difficult to identify.

These activities constitute the essence of the shift analyst's responsibilities. The remainder appear to be typing commands that the project analyst gives him. Consequently, the shift analyst position could probably be completely eliminated by automation.

II. DECISION MAKING IN KNOWN SITUATIONS

Currently, once the shift analyst identifies a trend or out of limit conditions, he reports it to the project analyst who decides what action to take. In general, the decisions the project analyst makes based on telemetry conditions appear quite simple. A single telemetry point is used to identify a spacecraft problem or situation and a single canned procedure is used to address it. This appears to be a very simple

If-Then situation--a single If condition (or two or three when spacecraft configuration must be taken into account) and a single Then consequent (e.g., use Proc 17 or contact experimenter X). Problem areas addressable by If-Then rules are frequently candidates for an expert system. An expert system could be used here but it would be pointless technically.

The rules are too simple, the reasoning chains are generally only one rule long, and consequently there is no room for exploration of reasoning that is not canned. The responding task could be dealt with much more efficiently from a technical point of view by avoiding the overhead of a production system and instead using standard software developed specifically for the application.

The main technical argument for using an expert system would be for the user interface. A good interface for an expert system would allow dynamic explanation traces and queries about system reasoning, old rules to be changed and new rules to be entered. As noted above, dynamic explanation would probably play little or no role here. In addition, rules could be manually changed or added in a conventional software system if the interface were properly designed. Automated generation and insertion of new rules will be discussed below.

III. DECISION MAKING IN NOVEL SITUATIONS

This is perhaps the most suitable area for incorporating artificial intelligence in ISEE operations. The task is currently performed by the project analyst who draws on knowledge of spacecraft (and sometimes experiment) design and operations, an understanding of the current configuration and orientation, and a knowledge of physics. When a solution is found, the project analyst generates a command sequence to effect it. If the situation is important enough, he will develop a canned procedure which can be used in the future to handle such situations. The procedure is entered into the system for ready access. In certain.cases the project analyst may have insufficient knowledge about a particular problem or subsystem and must call in an expert. The results of this analysis may or may not be codified in a procedure depending on whether the problem might ever arise again.

Artificial intelligence could be used to generate new procedures and command sequences to handle new situations. Such a Procedure Generation System (PGS) system would incorporate the same knowledge the project analyst (or expert) currently draws on: spacecraft design and operations, configuration, orientation, and physics.

It would have three analytic components. The first would be diagnostic. It would identify out of limit conditions and the implicated spacecraft configuration. It would determine reachable configurations that the craft could be safely moved to. The second would plan the sequence of changes necessary for changing to the one of those desirable configurations. The third would instantiate this change sequence as a sequence of commands, which could then be uplinked to effect the desired re-configuration. This final command sequence could be codified as a procedure to be used when similar situations arise in the future.

This PGS could also be used to generate all the procedures that would be used to manage a new spacecraft. It would only need to be told which situation to develop procedures for.

Fully automating this activity would eliminate the most interesting part of the project analyst's job and to some extent the reliance on additional outside experts. While this would be a significant undertaking, a demonstration of this capability for a particular subset of spacecraft operations would be tractable under the current contact.

IV. GROUND COVERAGE SCHEDULING

The problem of scheduling ground support coverage for the ISEE spacecraft decomposes to at least five activities, only two of which are performed wholly within ISEE. First the project analyst develops a seven day advance schedule request. This is essentially canned and is derived from a generic schedule, which is established in accordance with design and operation requirements. Occasionally it also incorporates special requests by experimenters and any special coverage needed by the project analysts for spacecraft operators.

This schedule request is submitted to the MSOCC schedulers who coordinate it with schedule requests for other spacecraft controlled out of MSOCC. In this coordinating, conflicts are resolved on the basis of priority, so ISEE may lose some coverage slots that were desired. The schedule request is then sent to NCC to be coordinated with all schedule requests handled by NCC. Coverage slots can again be lost.

The resultant seven day advance schedule is sent to the ISEE project analysts. If any of the specific unmet schedule requests were critical, the analyst contacts the NCC by phone and negotiates to get coverage at the same time, if possible, or at another time when the same objectives could be accomplished.

The final seven day advance schedule is only modified if coverage is missing for a critical period, another mission with higher priority enters a new conflicting request, or an emergency arises such as experimenter calling up with a request for coverage that he had not realized he needed.

The process of generating the seven day advance schedule involves a look-up to get the canned generic schedule with perhaps a few minor modifications (drawn from a small set of possible actions) by the project analyst or experimenter. This could be automated by a short conventional software program coupled with a simple menu driven interface for making modifications. The output could be transmitted to the schedulers electronically or printed for hand transmission.

The second phase of ISEE activity in scheduling is negotiating both with MSOCC schedulers and the NCC schedulers. These negotiations can get quite complicated. For classification reasons the NCC, for example, is frequently unable to suggest alternatives to the ISEE request. Thus the computer working the ISEE side would be required to generate alternatives. The algorithm used by the project analyst here seems straight-forward: identify the time interval in which the requested coverage could happen and still allow the desired actions to take place; then, vary the requested time within that interval in a fairly regular way, checking at each new choice point to assure that there is still an appropriate time slice for achieving the desired actions. An automated system could even maintain historical data if there were identifiable patterns of resource conflict. The bulk of this could be handled by standard software although an expert system could be used if the system were to do much reasoning about historical patterns. As it is uncertain whether these patterns exist, it is not clear such reasoning would be worthwhile.

There might be considerable resistance from the NCC side to having the ISEE half of the negotiations automated. NCC people would then be interacting with a computer system that had limited knowledge of the scheduling process and could not be led by intonation or innuendo to a suitable alternative. It may, however, hasten the automation of the NCC scheduling process after which the computers would be negotiating between themselves.

V. EXPERIMENT SCHEDULING

Experiment scheduling on ISEE is now almost entirely rote. Some experiments run all the time. Others are alternated in pairs--X on one day and Y off, Y on the next day and X off--due to conflict over spacecraft resources. Occasionally an experimenter requests special time on in what would normally be an off period. To comply, the project analyst may need to resolve any resource conflicts, which may mean taking another experiment off line. Clearly the rote scheduling could be automated with standard software. It appears that the resource conflict resolution is simple enough that special request processing could be handled with conventional software also. It would be possible to use some artificial intelligence for the special request processing, but the use would probably be quite trivial. The decision logic might be something like:

```
Is there any countervailing condition (e.g., resource conflict)?

        No:  Then schedule that experiment to be turned on for the
             appropriate period.

        Yes: (Assuming problem is resource conflict with another experiment)

             Will any critical data be lost by shutting off the conflicting
             experiment?
```

No: Schedule conflicting experiment off and requesting experiment on for appropriate period.

Yes: Then is the relative priority of the requesting experiment higher than that of the conflicting experiment?

- No: Refuse request

- Yes: Schedule conflicting experiment off and requesting experiment on for appropriate period.

Questions of relative priorities and crucialness of data would have to be addressed, but these would be given to the software system to manipulate so that it would not have to deduce them. This is in part a practical and in part a political matter.

VI. EXPERIMENT MANAGEMENT

Management of the ISEE experiments is currently done almost entirely by the experimenters themselves. They send the project analysts the command sequences they want sent to their experiments. Frequently they give the analysts these commands in their binary coded representation. Sometimes, for the analysts convenience, they include the command acronyms so that the analyst can verify that the commands will have not adverse effects on the spacecraft.

For another spacecraft the problem would presumably be different. An experimenter might give the project analyst a high level request and the analyst would determine what commands would have to be used to accomplish it. In any case the final command sequence is typed into the system by the shift analyst.

Automation could be introduced in at least two ways. The first would be to continue allowing (and in effect requiring) the experimenter to generate his own command sequences but allow him to type them directly into the system (eliminating the shift analyst's task) and send them directly to the already existing command management system (CMS) or a similar system for verification (thereby eliminating the project analyst's task). If the CMS were used, this could be done simply by granting the experimenter electronic access to the system, probably via a dial-up computer line. The problem of scheduling the actual uplink of the commands would, of course, still have to be addressed.

A second way automation could be introduced would be to develop a full Command Generation System (CGS). In this alternative the experimenter would not need to generate his own commands. He could instead specify a high level request such as "focus sensor A for my experiment on Alpha Piscis Austrini." The system would consider current sensor and satellite orientation and determine what would have to be changed to effect the request.

For each change required to fulfill the request, the system would evaluate the impact on both the experiment and the spacecraft as a whole. For every adverse effect, it would see if there were any other way to accomplish the same thing and if not, it would weigh the priority of the experiment request against that of the affected component or activity. If the priority of the request were not high enough, it would be denied. This type of reasoning could be used during a high level planning of the change sequence. Once the high level change sequence is established, it could be instantiated in a specific command sequence.

There are two approaches to instantiating the change sequences as command sequences for the CGS. The command sequence should be generated completely and then set to the CMS for validation and eventually uplinked. Alternately, the specific sequence would be generated using reasoning similar to that used to generate the higher level change sequence. Each command step would be evaluated as it was generated. Any problems would be identified as they arose rather than after the entire sequence--with its compounded problems--had been established. This would be considerably more efficient.

A command generation system which functioned in this way would be highly useful: as mentioned above, it would remove the involvement of both the shift analyst and the project analyst. In general, such a system would need a fair amount of knowledge about the spacecraft and its experiments.

In the case of ISEE, however, the range of things an experimenter could request appears to be very small, e.g., changing voltage level of the experiment or sensitivity level of the sensors between high and low, or focusing a sensor on a new object. Thus the amount of knowledge needed by such a system for ISEE may be fairly small. In any event, it appears that the reasoning (as distinct from the knowledge) required to accomplish command generation in general is relatively straightforward. As a result, a command generation system for the ISEE application may be quite simple and hence implemented as readily using conventional software as with an expert system approach.

VII. CONCLUSION

The objective of this effort is to build a demonstration system based on the ISEE mission operation support center activity that could be expanded towards space station application. For a more complicated mission, either a Command Generation System (CGS) or a Procedure Generation System (PGS) could be considered for this end. For the first of these a knowledge base of which requests and commands are possible given the constraint of a situation could be implemented as rules in a production system such as HAPS. The system would contain a model of the current spacecraft configuration and would use these two components to generate command sequences to accomplish requests.

A Procedure Generation System would have the same configuration model and a similar knowledge base containing information about handling novel situations as well as about sequencing commands.

Other aspects and components of the systems would be relatively similar. As an example, the man-machine interface for the CGS might allow more interaction (since more input would come from the human), but the style of interaction would be the same as the style of the PGS interface.

In terms of content as opposed to presentation, the PGS would address problems of spacecraft management while the CGS would be directed at experiment management. For suitably sophisticated missions, these problems may be equally complex. For ISEE, however, command generation for experiment management appears too simplistic for a real expert system and it seems that a Procedure Generation System would make the more interesting and sophisticated demonstration.

In addition to ISEE, we have examined the Dynamic Explorer (DE), Nimbus, International Ultraviolet Explorer (IUE), Solar Maximum Mission (SMM), and ERBS. ERBS proved unviable since its personnel would be heavily involved in prelaunch activities during 1984. Most of the others had also established rote operating procedures. A possible exception is IUE, which does more complex experiment commanding than the other missions: a re-orientation of a sensor frequently requires re-orientation of the spacecraft, which may make for involved planning.

We have identified a two-stage development process for a procedure generation system (PGS). A PGS has two components--a planner and a diagnostic unit--which would be developed in separate stages. The planner would take as inputs an initial spacecraft configuration and a final, target configuration. It would work hierarchically, planning the sequence of changes necessary to move from one to the other and then instantiating that change sequence in a final output as a sequence of commands which could be uplinked. The diagnostic unit, in contrast, would respond to undesirable states by accepting as input the initial (undesirable) configuration. The diagnostic unit would determine as a result of its analysis what a suitable final configuration would be. The initial (undesirable) and final (desirable) configuration would be given as input to the planner. During its analytic process, the diagnostic unit would also identify a series of changes required to move from the initial to the final safe or desired state.

These changes would probably be at a more abstract level than a change sequence established by the planner. The hierarchical planner, however, can use the abstract sequence of state changes from the diagnostic unit to generate the uplinkable, executable command sequence. This makes it possible to use the abstract change sequence (generated as output by the diagnostic unit) as an intermediate level input to the planner. The planner would use the abstract sequence as if the planner had generated it itself.

In summary, the two-phase development strategy is to develop the planner in the first stage and develop the diagnostic unit later. This would provide a demonstration system at the end of the first phase with the possibility of extending it. The diagnostic component would be significantly more sophisticated and may not be tractable under the scope of the present contract. It would, however, be a highly desirable addition to the system: an automated system for space station support would undoubtedly require such a capability.

APPENDIX E

Requirements Review

CONTENTS

I. INTRODUCTION

The Satellite Operations Support Expert System contract is an effort to identify aspects of satellite ground support activity at Goddard Space Flight Center (GSFC) which could profitably be automated with artificial intelligence (AI) and to develop a feasibility demonstration fo the automation of one such area. To accommodate the one year contract time frame, GSFC selected the International Sun Earth Explorer (ISEE) control facility as the application domain. The three satellites are relatively simple so that operational intricacies will not complicate the development process. This is to allow the resultant system to illustrate the application and development of AI technology in the satellite support environment.

Investigation of the ISEE operations center revealed six major areas of activities: 1) telemetry monitoring (for both experiments and spacecraft operations), 2) deciding what action to take in response to known problems, 3) deciding what action to take for novel problems, 4) ground coverage scheduling, 5) experiment scheduling, and 6) experiment management. These are discussed in detail in the documentation on application choices (the deliverable for task one) which should be referred to for more information.

That report indicated that almost all the activities in each of the areas could be automated using conventional software. Procedure generation and command generation are two exceptions that were identified in that document. When these were probed more thoroughly it became clear that both of them could be automated in two entirely different ways. One way would use the shallow reasoning approach of a standard rule-based system. However, the rules here would be quite simple and there would be little if any overlap between the rules required to generate one procedure (or command sequence) and those required to generate another. The result would be extremely hardwired and have very little substance. It could be achieved readily with conventional software.

Another approach to automating the procedure or command generation process would be exceedingly difficult. It would use the causal or deep reasoning techniques in artificial intelligence and would require the most detailed, intricate model available of the structure and function of every aspect of the spacecraft. In addition it would have to have considerable knowledge of physics, astrophysics, electronics and the procedure/command language. It would have to incorporate a robust, diverse set of problem solving strategies and methodologies. This would be, in short, a complicated project which should be done during the design of a spacecraft: retrofitting such a system would give relatively little benefit. The ultimate payoff of this type of system would depend on the longevity and versatility of use of the spacecraft.

It is possible that AI could play a role in the executive control functions of a fully automated control center. Had various activities such as monitoring altering and scheduling, been automated, some form of executive control would be required. The exact nature of this control environment is not clear. An in-depth operations analysis is needed to determine what control can be achieved through conventional software and what will require artificial intelligence. This analysis will be quite involved and is not addressed under this effort. We recommend that a thorough operations analysis be pursued. We further suggest that any future spacecraft and control centers be designed to incorporate all feasible automation (conventional as well as AI) to existing systems.

AI may be able to play a role in high level mission workarounds. In the event of a significant failure of some subsystem or component, it may become impossible to achieve all spacecraft missions. Executing and scheduling of the missions may need to be replanned in accord with the reduced resources of the degraded spacecraft. This replanning could be accomplished by using conventional operating systems scheduling algorithms or, perhaps, using planning techniques developed in artificial intelligence.

The other area where artificial intelligence could be useful is in the fault handling (FH) arena. For some but not all subsystems, FH can pose a complex problem. There are three approaches to FH, through only two are in general use. The first is through hardware. Hardware is often designed to be self-correcting: to identify failed componentry and work around it, perhaps by switching to redundant components or systems. Likewise, on-board sensors can be used to detect a fault which is then handled from the ground. The weight of this hardware is an issue, however. Greater weight means greater cost in resources for launching and controlling a vehicle. Thus, for example, while twelve sensors might give complete information about a particular subsystem, only four might actually be used. Whether this weight/efficiency compromise continues in the future will depend on advances in electronics technology for reducing the size and weight of sensors, chips and the like. With sufficient advances, fault handling may be done predominantly in hardware on-board: much of the capability appears to exist at this time.

The second commonly used means of fault handling is human troubleshooting from the ground. Generally used in conjunction with on-board hardware, this can involve the spacecraft analyst and, if the fault or subsystem is complicated enough, an outside expert. This can result in a considerable technical support staff for the maintenance of the spacecrafts.

Another, though apparently little used, possiblity is software automation. This can frequently be accomplished with conventional software: some control centers have detailed flowcharts specifying the algorithm for isolating faults in various subsystems. In these centers fault symptoms have been thoroughly identified. These symptoms act as cues for using one of the algorithms. A few subsystems appear to be more complex and may merit the use of AI technology. It must be stressed, however, that the utility of any software automation of fault handling must be assessed in light of the possibilities for performing the function on board in hardware.

For this contract, we have proposed developing a demonstration fault handling system for a hydrazine propulsion system (HPS). This appears to be a rather generic problem. Hydrazine systems are the most common monopropellant blowdown propulsion system (MBPS) and MBPSs are one of the most common propulsion mechanisms on spacecraft. The structure of MBPSs is quite uniform across spacecraft. Despite their lower efficiency, hydrazine systems will be used on future spacecraft, due to their economy and weight, and are a likely candidate for use in space station. Finally, the HPS is one of the more complex subsystems on ISEE and should be sufficient to illustrate some AI concepts.

II. General Introduction to Hydrazine Systems

Introduction: Hydrazine systems are the most common monopropellant blowdown systems in use on spacecraft today. Hydrazine systems have been used on a number of spacecraft at Goddard Space Flight Center (GSFC), including the International Sun Earth Explorer (ISEE). As mentioned above, further use of hydrazine systems seems assured.

Reasons for the common use of hydrazine include:

- o It is a storable liquid monopropellant.
- o It is compatible with many materials.
- o It is shock and friction insensitive.
- o It is a stable liquid (safely heated to 500°F)
- o It has safe, simple handling procedures.
- o It is an inexpensive fuel.
- o It is flight tested.
- o It can withstand long and short jet firings.
- o It does little or no damage to scientific equipment.

Purpose of Hydrazine Systems: The general purpose of hydrazine systems is to perform attitude control functions. The functions performed by the hydrazine system on, for example, IUE are:

- o Notation control.
- o Control of the precession of the spin axis.
- o Spin of the spacecraft.
- o Spacecraft torquing to acquire the proper sun angle in the event of attitude loss.
- o Velocity correction to initially acquire the orbit station.
- o Velocity correction for east-west station keeping (Delta-V).
- o Spacecraft torquing to unload reaction wheels.

Properties of Hydrazine: Some chemical and physical properties of hydrazine are:

- o The chemical equation for hydrazine is N_2H_4.
- o Hydrazine is a colorless liquid with physical characteristics similar to water.
- o Molecular weight 32.
- o Freezing point 34.5°F.
- o Boiling point 235.4°F.
- o Specific gravity 1.008 @ 68°F.
- o Vapor pressure 0.2 psia @ 68°F.
- o Hydrazine decomposes into hydrogen and nitrogen at high temperatures (at slightly lower temperatures hydrazine decomposes into hydrogen, nitrogen and ammonia NH_3).

Functioning of Hydrazine Systems: Hydrazine systems function by pushing hydrazine fuel from storage tanks down to a series of thrusters. The fuel lines are interconnected by a series of latch valves designed to minimize the effects of valve failures. The hydrazine fuel, when it reaches an engine, passes over a heated catalyst bed. The hydrazine expands to form a hot gas. This gas is expelled out the engine nozzle providing the thrust used in performing some maneuver (see Figure 1).

Components making up a hydrazine system: The hydrazine system to be used in the demonstration (shown in Figure 2) contains:

- Eight propellant tanks.
- Fill/vent valves for both groups of four tanks.
- Fill/drain valves for both groups of four tanks.
- Three pressure sensors.
- Six fuel line filters.
- Seven latch valves.
- Twelve heaters.
- Twenty-four temperature sensors.
- Twelve catalyst beds.
- Twelve engine valves.

Propellant tanks: The propellant tanks store the hydrazine fuel used throughout the life of the spacecraft. The propellant tanks are usually grouped pairwise in order to maintain the spin balance of the craft.

There are two basic designs for the fuel tanks: 1) a bladder system and 2) a gas propellant system. ISEE has the bladder system. In the bladder system a thin membrane separates the hydrazine from a pressurizing gas (usually He). The pressure in the tank is raised by filling the pressurant side of the bladder with the pressurizing gas. The gas propellant system works basically like a spray can. The pressurizing gas is injected into the tank with the hydrazine fuel. The gas and the fuel do not mix and the pressurant forces the hydrazine fuel along the sides of the fuel tank and through a filter designed to keep the pressurant and fuel separate.

Fill/vent valves: The fill/vent valves are used to fill and vent the pressurant to the fuel tank. The fill/vent valve is used before the mission to fill the fuel tank with pressurant.

Fill/drain valve: The fill/drain valves are used to fill the fuel tanks with hydrazine before launch.

Fuel lines: The fuel lines connect all components making up the propulsion system. To reduce weight, the fuel lines are made of titanium with stainless steel joints.

Fuel filters: Filters are placed in the fuel lines to prevent particles from interfering with the function of the latch valves, engine valves and catalyst beds. Filters are located after each fuel tank and before each valve. If a particle reaches either an engine valve or a latch valve it could prevent the valve from closing properly. This would result in fuel leaking through the seat of the valve.

Latch valves: Latch valves separate sections of the fuel lines and fuel tanks. Latch valves are commanded to be in one of two states: open or closed. Once commanded a latch valve stays in the commanded state until it is given another command (e.g., if a latch valve is commanded to open it will remain open until it is commanded to close).

Heaters: Solar heating is intended to be the primary source of heating for HPS. Electrical heaters are provided to the fuel tanks and to the fuel lines to assure that the hydrazine fuel does not freeze resulting in damage to the system. Heaters are also provided for the catalyst beds in order to maintain temperatures that allow the catalyst to act more efficiently.

Engine valves: The engine valves are located before the catalyst bed of each thruster. It is through these valves that the firing of the thrusters is controlled. The engine valve is a solenoid which is commanded to open. Once commanded the engine valve opens for a short period of time and then closes. Thus, commanding an engine to maintain its thrust requires sending a stream of open commands to the engine valve.

Pressure sensors: Pressure sensors are located after each set of fuel tanks. These pressure guages have an accuracy range of ± 1%.

Catalyst bed: The catalytic agent in the catalyst bed is Iridium (Shell 405). The catalyst bed is fed with hydrazine through a cluster of capillary tubes connected to the fuel line.

Heat sensors: Heat sensors are located on each fuel tank and after each latch or engine valve. There is also a heat sensor on each catalyst bed. The purpose of the sensors on the fuel tanks and valves is to monitor the temperature of the hydrazine fuel. The temperature sensors on the catalyst beds allow for monitoring the catalyst bed temperature to assure that they are operating efficiently. The heat sensors have an accuracy range of ± 2%.

Telemetry data: There are a number of telemetry streams giving information about the health of the hydrazine system:

- Fuel line pressure data. A numeric value representing the pressure in the fuel line.
- Latch valve sensor data. A value for each latch valve showing if the valve is opened or closed.
- Engine valve sensor data. A value for each engine valve showing if the valve is opened or closed.
- Heater temperature gauge data. A numeric value for each sensor which shows the temperature at that point in the system.

- <u>Catalyst bed temperature data</u>. A numeric value showing the temperature of a catalyst bed.

- <u>Engine valve temperature data</u>. A numeric value showing the temperature of an engine valve.

- <u>Attitude data</u>. Data reflecting the position of the spacecraft. If less or greater than expected, this data may signal a malfunction in the propulsion system.

III. Faults and Their Workarounds

We will handle hydrazine system failures which correspond to the failure of an individual system component. Failures in the system must be inferred from telemetry data received from the spacecraft.

Failures that could occur in a standard hydrazine propulsion system are:

- o Leak in the hydrazine storage tank.
- o Leak in a section of the line.
- o Failure of a latch valve to open.
- o Failure of a latch valve to close.
- o Failure of an engine valve to open.
- o Failure of an engine valve to close.
- o Failure of a heater to turn off.
- o Failure of a heater to turn on.
- o Failure of a catalyst bed.

Some of the faults listed below have overlapping symptoms. Also, in some cases it may be impossible to narrow a fault down to a single component. For example, a clogged filter and a failing catalyst bed can give the same symptoms.

Faults, symptoms and their workarounds.

FAULT: Leak in the hydrazine fuel tank.

SYMPTOMS:

- o Loss of pressure in the fuel line.
- o Engine thrust results in obtaining less attitude correction than expected.
- o Catalyst bed and engine valve temperatures lower than expected.

WORKAROUND: Vent the fuel outside the spacecraft by using the thrusters. The thrusters are used to prevent the fuel from harming the spacecraft.

FAULT: Leak in a section of the fuel line.

SYMPTOMS:

- o Engine thrust results in obtaining less attitude correction than expected.
- o A lower pressure in the fuel lines than expected.
- o A lower pressure in the fuel tanks.
- o Catalyst bed and engine valve temperatures less than expected.

WORKAROUND: The workaround is to shut off the part of the system containing the leak. This is done by closing the latch valve, or latch valves, immediately above the leak.

FAULT: Failure of the upper-level latch valve to close.

SYMPTOMS:

- o Engine thrust results in obtaining more attitude correction than expected (if lower level latch valves are open).
- o Sensor for the latch valve indicates the valve is open.
- o Catalyst bed and engine valve temperatures greater than expected (if lower-level valves are open).

WORKAROUND: The workaround is to shut the latch valve immediately above this latch valve. Fuel can be sent through this line with some risk of causing a pressure spike (or water hammer) when hydrazine passes through the open valve. A pressure spike could damage the valve, pressure sensors or fuel tank bladder membrane.

FAULT: Failure of the lower-level latch valve to close.

SYMPTOMS:

- o Sensor for the latch valve indicates the valve is open.

WORKAROUND: The workaround in this situation is to operate the system normally as long as the fuel in the region is intact.

FAULT: Failure of a upper-level or lower-level latch valve to open.

SYMPTOMS:

- o Engine thrust results in obtaining no attitude correction or less than expected.
- o Sensor for the latch valve indicates the valve is closed.
- o Low temperature for the catalyst bed and engine valve.

WORKAROUND: The effect of this fault is to functionally cut off the section of the system below the stuck valve. If a redundant fuel line is available it should be used. If a redundant fuel line is not available this part of the system has lost its functionality.

FAULT: Failure of the engine valve to close.

SYMPTOMS:

- o Engine thrust results in obtaining more attitude than expected (thrusters will fire as long as there is an open line to one of the fuel tanks).
- o Sensor indicates that the engine valve is open.
- o Temperatures for engine valve and catalyst bed higher than expected.

WORKAROUND: Shut off the latch valve immediately above the engine valve. Use a redundant set of thrusters. If a redundant set of thrusters is not available this set may be used with a greatly reduced control over the thrust.

FAULT: Failure of the engine valve to open.

SYMPTOMS:

- o Engine thrust results in obtaining no attitude correction or less than expected.
- o Sensor for the engine valve indicates the valve is closed.
- o Low temperature in the catalyst bed and engine valve (i.e., the thruster is not firing).

WORKAROUND: This fault makes it impossible to use the thrusters associated with this engine valve. Repeated attempts to command the valve open may eventually succeed. Otherwise a redundant set of thrusters must be used.

FAULT: Failure of fuel-line heater to turn off.

SYMPTOM:

- o Heater sensor indicates high temperature.
- o Engine thrust results in obtaining less attitude than expected (if the temperature of the line is great enough to break the fuel down into hydrogen, nitrogen and ammonia).

- o Low temperature in the catalyst bed and engine valve (this is the result of the thrusters not firing efficiently due to the breakdown of the fuel).

<u>WORKAROUND</u>: Do not use this part of the system as long as the temperature is high enough to break down the hydrazine. If the hydrazine decomposes into its component gases the efficiency of thruster burns will be reduced because of gas bubbles in the fuel lines.

<u>FAULT</u>: Failure of a fuel-line heater to turn on.

<u>SYMPTOMS</u>:

- o Heater sensor indicates low temperature.
- o Engine thrust results in obtaining less attitude and catalyst bed and engine valve temperature lower than expected. This loss of thrust is due to hydrazine freezing in the fuel line and either completely clogging the line or restricting the flow of fuel.

<u>WORKAROUND</u>: If the line does not freeze, then no workaround is needed. If the line does freeze, two workarounds are possible: 1) bathe the spacecraft in the sun to heat the fuel line or 2) use a redundant part of the system.

<u>FAULT</u>: Failure of catalyst-bed heater to turn off.

<u>SYMPTOM</u>:

- o High catalyst bed temperature.
- o High temperature in the engine valve.
- o Lowered engine thrust: the result of hydrazine breaking down before the engine valve and mixing gas with the hydrazine fuel.

<u>WORKAROUND</u>: If the temperature gets too hot then heat could transfer across the manifold causing the hydrazine fuel to break into its component gases. This will degrade the performance of the thruster. The workaround would be to use a redundant set of thrusters, if they are available. If the redundant set of thrusters is not available, then the degraded performance will have to be accepted.

FAULT: Failure of a catalyst-bed heater to turn on.

SYMPTOMS:

- o Low catalyst bed temperature.
- o Engine valve temperature could be lower than expected and engine thrust could result in obtaining less attitude than expected. The lower temperature and loss of thrust is due to hydrazine not igniting as efficiently as it would at a higher temperature.

WORKAROUND: Two workarounds are possible: 1) bathe the spacecraft in the sun to heat the catalyst bed or 2) use a redundant part of the system.

FAULT: Failure of the catalyst bed.

SYMPTOMS:

- o Attitude change less than expected.
- o Low temperature in the catalyst bed (due to lower reaction rate of the hydrazine).

WORKAROUND: Use redundant set of thrusters, if possible. If a redundant set of thrusters is not available then the reduced thrust must be accepted as a permanent characteristic of the system.

Other system failures which could mimic a hydrazine failure: Some symptoms of hydrazine failures which could be mimicked by other systems on spacecraft are:

- o Sensor failure (e.g., the sensor could represent a valve as being closed when in fact it is open).
- o On board computer (OBC) malfunction (e.g., the OBC could send down incorrect telemetry data). (Not an ISEE problem.)
- o Gyro failure (results in unexpected attitude change). (Not an ISEE problem.)
- o Wheel driver assembly (result in an unexpected attitude change). (Not an ISEE problem.)

IV. System Configuration

The proposed system contains five major modules, as illustrated: an executive, I/O, subsystem model, telemetry generator, and fault handler. The conceptual flow through the system is as follows: the I/O module presents to the user a menu of options which he uses to specify the type and location of a fault. For any given test or demonstration the system will be capable of handling one fault. The executive passes this to the model module, which adjusts the internal representation of the HPS appropriately. The telemetry generator then uses this internal representation to produce telemetry which is passed to the fault handling component. The latter uses its isolation techniques to locate the failure and determines a method of handling it. This will generally involve changing the state of subsystem components in the model, e.g., opening and closing latch valves and observing the response. These changes are presented to the user textually and are reflected in normalized telemetry data. Once identified, the nature of the fault and its workaround are presented to the user.

Executive: The executive monitors and coordinates the overall flow of control through the system.

I/O: The I/O module is responsible for the man-machine interface. It allows the user to specify the type and location of a fault and displays this on the screen. As the fault handling module progressively localizes the failure, the user will be given a description of the isolation process. Once the fault is located its nature and workaround will be presented to the user.

Model: The model component maintains an internal representation of the structure and function of the HPS. It modifies the current status of the model in accord with the faults specified by the user and the probing actions and workaround suggestions of the fault handling module.

Telemetry Generator: The telemetry generation module provides simulated telemetry data to the fault handling component. From its understanding of how telemetry source points vary under normal and faulty conditions, this produces telemetry data appropriate to the current state of the HPS, as indicated by the internal model.

Fault Handling Module: The fault handling module (FHM) detects and isolates failures based on the telemetry data produced by the telemetry generator, and identifies workaround strategies. The FHM is the only portion of the system that uses artificial intelligence technology.

Telemetry data is processed by a conventional monitoring and alerting submodule for detecting exception events, i.e., faults. The remainder of the module's processing follows the standard production system format. Indications of the identified failures are inserted into "working memory". The "rule base" encodes a straightforward algorithm that is used for isolating failures. The workaround emerges as an immediate consequence of having isolated the fault: there is, in general, a single option.

The process of locating the fault generally involves opening and closing valves at various points along the hydrazine lines. These changes are effected in the model, which results in changes in the telemetry data produced by the telemetry generator. The new telemetry data is fed back to the FHM which can then assess the consequence of its probing action and can choose the appropriate part of the algorithm to use next.

As the FHM probes, it progressively eliminates portions of the HPS as candidates for containing the failure. This reduction in search space can be presented textually to the user's screen by the I/O module.

V. Summary

The purpose of this Satellite Ground Support Expert System contract is to illustrate some artificial intelligence concepts in the satellite ground support domain. Fault handling has been identified as an aspect of ground support which might benefit from automation via AI. A hydrazine propulsion system has been chosen as the application for fault handling as it appears sufficient for the above purpose. We will be able to demonstrate some simple aspects of rule based systems, which will account for 15-20% of the code for the entire demonstration system. We intend to proceed with the development of the system for demonstration by November 1984.

APPENDIX F

Software Design Documentation

Introduction

Under the Satellite Operations Support Expert System, we are developing a demonstration system to perform automated fault handling for hydrazine propulsion systems (HPS). This system consists of six major components: the executive, the I/O module, the exposure module, the propulsion system model, the telemetry generator, and the fault handler. All but the last of these--the fault handler--are considered supporting software and will be documented in this report. The fault handler is effectively the knowledge base of this system, and in accordance with the contract, will be documented under a separate cover.

System Overview

The overall flow of control through the program is directed by the executive. After initialization of the program environment, the executive is invoked. It calls a part of the I/O module which queries the user for a choice of propulsion system configuration, fault type, and fault location. Choice of configuration entails identifying or constructing a model of the propulsion system for internal use by the system. If the user selects the configuration from ISEE or IUE, the appropriate model is loaded from a file. Otherwise, it is constructed as the user specifies the components and structure. When the fault is selected, the system's internal model of the HPS is modified accordingly for use by the telemetry generator and fault handler.

The executive then calls the exposure module which examines the fault selected in the context of the configuration and determines what state the propulsion system should be in to cause the fault to influence its operation. This is a state in which symptoms will appear in the telemetry stream as abnormal or unexpected values. The exposure module identifies how the HPS should be "commanded" to move into that state and returns a set of command actions to the executive.

The executive processes these actions by modifying the internal model appropriately, informing the user of the actions taken, and calling the telemetry generator. As discussed below, the telemetry generator produces telemetry to reflect the current state of the HPS. The telemetry is displayed to the user and examined for unexpected values by a distributed monitoring system embedded in the telemetry generator. Any unexpected values are reported to the fault handling module (FHM) both quantitatively (e.g., "catalyst bed temperature 850°") and qualitatively (e.g., "catalyst bed temperature is higher than expected").

The FHM is a rule-based module which, as mentioned above, will be documented under separate cover. Briefly, it identifies candidate pathways through the HPS, on which the fault might lie and searches those pathways, commanding valves to open and close and thrusters to fire. It draws conclusions about the location of the fault from the consequences of these actions, i.e., from the telemetry values produced after the action is effected. Once the fault is located, the FHM identifies and effects a workaround. Where appropriate, these actions and their motivations are described to the user.

When the workaround has been implemented and telemetry has normalized, the user is given three options:

a) to end the session;

b) to run another demonstration on the same or different configuration beginning from normal fault-free HPS operation;

c) to run another demonstration on the same configuration, beginning in its current, "degraded" state--retaining the original fault and its workaround and adding a second fault.

In a), the program terminates. In either b) or c), once the configuration and fault are known, the process proceeds as above.

The Executive

The executive is a straightforward controlling program which feeds information between subordinate modules. It is invoked to coordinate each trial, allowing for reinitialization between trials as appropriate. It first calls the I/O module to establish the user's choice of HPS configurations, fault type, and fault location. It uses the information returned to modify the internal HPS model. It passes the same information to the exposure module which determines how to expose the fault, i.e., how to "command the spacecraft" so that symptoms of the fault will appear in the telemetry. The exposure module returns a set of command actions which the executive processes by 1) modifying the model (i.e., changing the state of the HPS); 2) informing the user; and the 3) calling the telemetry generation module.

After producing and displaying new telemetry values, the telemetry generator returns a list of unexpected values. The executive sends these to the fault handling module. At various points, the FHM will send back to the executive commands to alter the state of the HPS. The executive handles these as it does the commands from the exposure module: modifying the model, informing the user, and calling the telemetry generator. New telemetry is produced and the results cycled back to the fault handler, by the executive. When the fault has been located and a workaround effected, the executive returns control to the initializing program.

The Input/Output Module

The I/O module provides the means whereby the user interacts with the demonstration system. There are three primary vehicles for this: menus, graphics, and textual presentations.

Menus are used for input to the systems. The configuration-choice menu allows the user to choose the HPS configuration from ISEE or IUE or to create a new one. In the latter case, a series of menus allow the user to specify the number of fuel tanks, latch valves, engine valves, and catalyst beds along with their structural relationships.

The fault-type menu presents a choice of the different kinds of faults. Based on the type of fault selected, the fault-location menu presents the choice of locations for the fault. At the end of a trial, the user will be given a menu with three options, as discussed above:

a) to end the demonstration sequence,

b) to begin afresh with an HPS (of whatever configuration) in normal operation and a new fault, or

c) to continue with the same configuration in its degraded (faulted) state, with workaround in place, and to select a second fault.

Elementary graphics using the VT100 terminal's "graphics characters" are used to display the HPS configurations of ISEE and IUE. The pictorial representation conveys all essential attributes of these configurations, including temperatures, pressures, and the open/close state of valves. Numerical telemetry values for components are displayed by the component location on the screen. It was decided not to use a high functionality graphics package due to potential difficulties in coordinating its use on the demonstration computer. Graphics representations for the HPS of ISEE and IUE are stored in disk files. In the event that the user elects to construct a new configuration, a graphic representation is not constructed. In this event, telemetry will be displayed in tabular form.

When ISEE or IUE configurations are used, textual descriptions and explanations of system activity will be presented at the bottom of the screen in a four-line scroll region. Most menus will be displayed in this region also. For user-created configurations, telemetry will be presented in one half of the screen with menus and text in the other. This permits the telemetry and/or graphics to remain unperturbed in the upper region of the same terminal screen as the text. The alternative of using multiple screens, i.e., one for graphics and another for menus and text, was deemed too unwieldly.

The Exposure Module

In actual operation of a propulsion system, it may be some time before a failed component is used and hence, before the existence of a fault is realized by the group support personnel. For the purpose of this demonstration, we have chose to expose the fault quickly, generating symptomatic evidence which the fault handling component can use for locating the failure.

The exposure module receives from the executive a description of the user-specified fault. From the type of fault that is chosen (leak, stuck valve, etc.), the exposure module determines what state the spacecraft must be in to cause symptoms to appear in the telemetry. It might be necessary, for example, to open a particular latch valve to expose a fuel line leak or to attempt to fire through a particular thruster when its catalyst bed is broken. Knowing the current state of the spacecraft, it can then establish what action or series of actions will put the spacecraft in the desired state. These can include opening or closing valves, firing a thruster, or in certain cases, doing nothing at all (the fault could be noticed while the system is "at rest"). The location of the fault may be checked to determine exactly where the action should be performed, but in some cases, the site of the exposure action or actions is independent of the fault location.

There are instances when a fault can be exposed under more than one set of conditions, and the symptoms displayed by exposing the fault will be different for each instance. In these cases, the exposure module randomly chooses which of the possible sets of exposure actions to execute. The list of actions to be performed is then passed back to the executive. The executive "performs" these actions by making appropriate modifications to the internal model.

Internal Model of the HPS

The system's internal model of the propulsion system is couched in a semantic network. Major components--valves, tanks, etc.--are conceptual nodes. Their structural relationship is indicated by connectivity links between the nodes. In addition, each component node has a series of other links indicating attributes such as its telemetry value(s) and whether the user has specified a fault for the component. Conceptually, this is also a frame structure where each major component is represented as a frame with various attribute-value pairs.

For ISEE and IUE configurations, this model is initialized from disk files to a fault-free operating state. For user-configurations, the model is built incrementally in accord with the user's specifications. In either case, when the user selects a fault for a particular component, the value of the fault-state attribute of the frame (node) corresponding to that component is changed to indicate that fault.

Similarly, commands are made to the HPS by the exposure module and the fault handler. The commands are actions such as opening and closing valves and firing thrusters. These actions are effected by changing the value of the operating-state attributes of the relevant HPS components.

The fault-state attribute along with the connectivity links and the operating-state attributes are used by the telemetry generator to produce telemetry appropriate to the current state of the HPS. New telemetry values are stored in the appropriate attribute slot in the model.

The only pieces of information that go from the model to the fault handling module are the connectivity relations at initialization time and later, at run time, the telemetry stream from the spacecraft. The FHM has no access to any fault state information except as reflected in the telemetry values. There are, conceptually, two internal models--one used by the I/O module and telemetry generator (which must know about fault states, etc., to simulate valid telemetry) and another used by the fault handler. For computational efficiency, the two are implemented in a single model.

Telemetry Generation

Function. The telemetry generation module serves two purposes in the fault handling system: 1) it computes telemetry values displayed for the user, and 2) it creates a qualitative difference list for the exposure module as described below.

There are four types of values representing the state of the hydrazine propulsion system: 1) telemetry values, 2) expected values, 3) actual values, and 4) qualitative values. Telemetry values: these are the values which are normally shown to ground control monitors. These values represent the state of the system as reported by the sensors aboard the spacecraft (whether these sensors are functioning properly or not). Expected values: these are values that would appear in the telemetry stream if all the components making up the system were functioning normally. Actual values: the actual values represent the true state of the system. The actual values are only used by the telemetry generation module. Qualitative values: these are derived from the telemetry values and the expected values. If the telemetry value for a component is greater that its expected value, then the qualitative value for this component is the value +1. However, if the telemetry value is lower than the expected value for a component, the component is assigned the qualitative value of -1.

Call structure. The telemetry generator is called by the executive module. The telemetry generation module is passed a list of commands (HPS components and commanded states for these components). There may be, for example, a command to open latch valve three. This command information allows the telemetry generation module to produce telemetry reflecting the current state of the HPS.

The telemetry generation module calls the I/O module which updates the telemetry displayed for the user. The changed telemetry values presented to the user reflect changes in the system as a ground station would see them. Once the I/O module updates the telemetry values on the user's display, it returns control to the telemetry generation module.

The telemetry generation module returns a list of qualitative differences to the executive module. This list of qualitative differences is not the complete list for the system, but rather the list of telemetry values which differ from their expected values.

General algorithm. The high-level algorithm for the telemetry generation system is given below:

1) Change the state of the components as commanded by the executive module.
2) Compute the telemetry values.
3) Compute the expected values.
4) Compute the actual values.
5) Construct the qualitative difference list.
6) Call the display routine to update the displayed telemetry values.
7) Return control to the executive module.

Generating telemetry. Given a simple hydrazine propulsion system (see figure below), the telemetry generation module computes telemetry by 1) computing the telemetry values for the current component, 2) calling routines to compute the telemetry for lower connected components, and 3) as control returns to the higher component, propagating the lowest fuel pressure value up the system. The telemetry values are generated by going depth-first through the hierarchy of connected components.

	TANK			
LV1		LV2		(latch valves)
EV1	EV2	EV3	EV4	(engine valves)
CB1	CB2	CB3	CB4	(catalyst-beds)
TH1	TH2	TH3	TH4	(thrusters)

For example, the call structure for the components in the above system would be updated to the values for: 1) TANK, 2) LV1, 3) EV1, 4) CB1, 5) TH1, 6) EV2, 7) CB2, 8) TH2, 9) LV2, 10) EV3, 11) CB3, 12) TH3, 13) EV4, 14) CB4, 15) TH4.

Assumptions. A number of assumptions were made in order to model a general hydrazine propulsion system. These relate to 1) the configuration of the spacecraft, 2) the environment outside the spacecraft, 3) the physical attributes of the faults.

The configuration of the spacecraft is an issue which, for the purposes of generality, we have basically left out of the model. There are a number of ways in which the architecture of a spacecraft could influence the behavior of the hydrazine propulsion system. As an example, the distance of the fuel lines from the outside of the spacecraft and the material on the outside of of the spacecraft (whether it is reflective or absorbent to light) will influence how heat dissipates from the fuel lines.

The environment is also simplified in the telemetry generation module. The outside environment is assumed to be 20°F. This prevents the telemetry generation module from having to know which parts of the spacecraft are in the sun and which parts are in the shade. This means that the telemetry generation module does not have to vary its model of heat dissipation from the fuel lines.

Certain physical attributes of faults have been simplified. The size of a leak is considered uniform for all leaks. The differences in fault isolation for small and large leaks are only in the time it takes for the fault to become noticeable (unless the leak was so small as to be undetectable). Another assumption is how the fuel lines and catalyst beds gain and lose heat. The basic model used for both the fuel lines and catalyst beds is a linear model of heat loss where the rate of change in temperature is affected by: 1) whether the temperature has reached one of the extremes in its range, and 2) whether the heater for the component is on. In the case of the fuel lines, the temperature can vary from 20°F to 80° F (the maximum heat the heaters can maintain).

Other quantities are derived from the physics of the system. For example, the volume of escaping fuel and ts effect on the pressure can be computed from a form of Bernolli's equation.

APPENDIX G

Knowledge Base Documentation

KNOWLEDGE BASE DOCUMENTATION

Introduction

The fault handler module (FHM) is the part of the expert system which makes the most use of artificial intelligence techniques. It interacts with the knowledge representation used by other parts of the system, reasons about the data it is given, requests that certain actions be performed when it requires additional data, and searches through pathways in the hydrazine propulsion system (HPS) to isolate the fault. Once that fault is found, the FHM uses its knowledge about the effects of various faults and the structure of the HPS to determine a "workaround", which is a series of actions designed to minimize or eliminate any decrease in performance of the HPS. The FHM implemented in this expert system consists of a knowledge base (sometimes referred to as a rule base) with various procedures which are used by the knowledge base.

A knowledge base makes use of several types of knowledge. Some knowledge is static, such as physical laws or structural relationships. Other knowledge may be dynamic, such as a temperature, a resource quantity, and other values which fluctuate with time. The knowledge can be explicitly stored as a table or list of data, or it can be implicitly stored in the structure of rules and procedures. The knowledge base of this FHM utilizes three areas of knowledge:

- o Knowledge of the structure of the HPS.
- o Knowledge of what symptoms may be produced by a fault.
- o Knowledge of how an action should affect the values from a sensor.

Knowledge of the structure of the HPS

This area of knowledge includes information about the number of each type of component and how the components are connected (i.e., which components are directly above and directly below a given component). Some basic assumptions are made about the possible configurations for an HPS, based on the structure of ISEE-III, IUE and other typical systems. These dictate what types of components may be above and below a specific type of component, and the presence of redundancy in the system.

The configuration is constrained by the following requirements:

Component type:	Component above:	Component below:
Tank	None	LV(s)
Latch valve (LV)	Tank(s) or LV(s)	LV(s) or EV(s)
Engine valve (EV)	LV(s)	CB *
Catalyst bed (CB)	EV *	Thruster *
Thruster	CB *	None

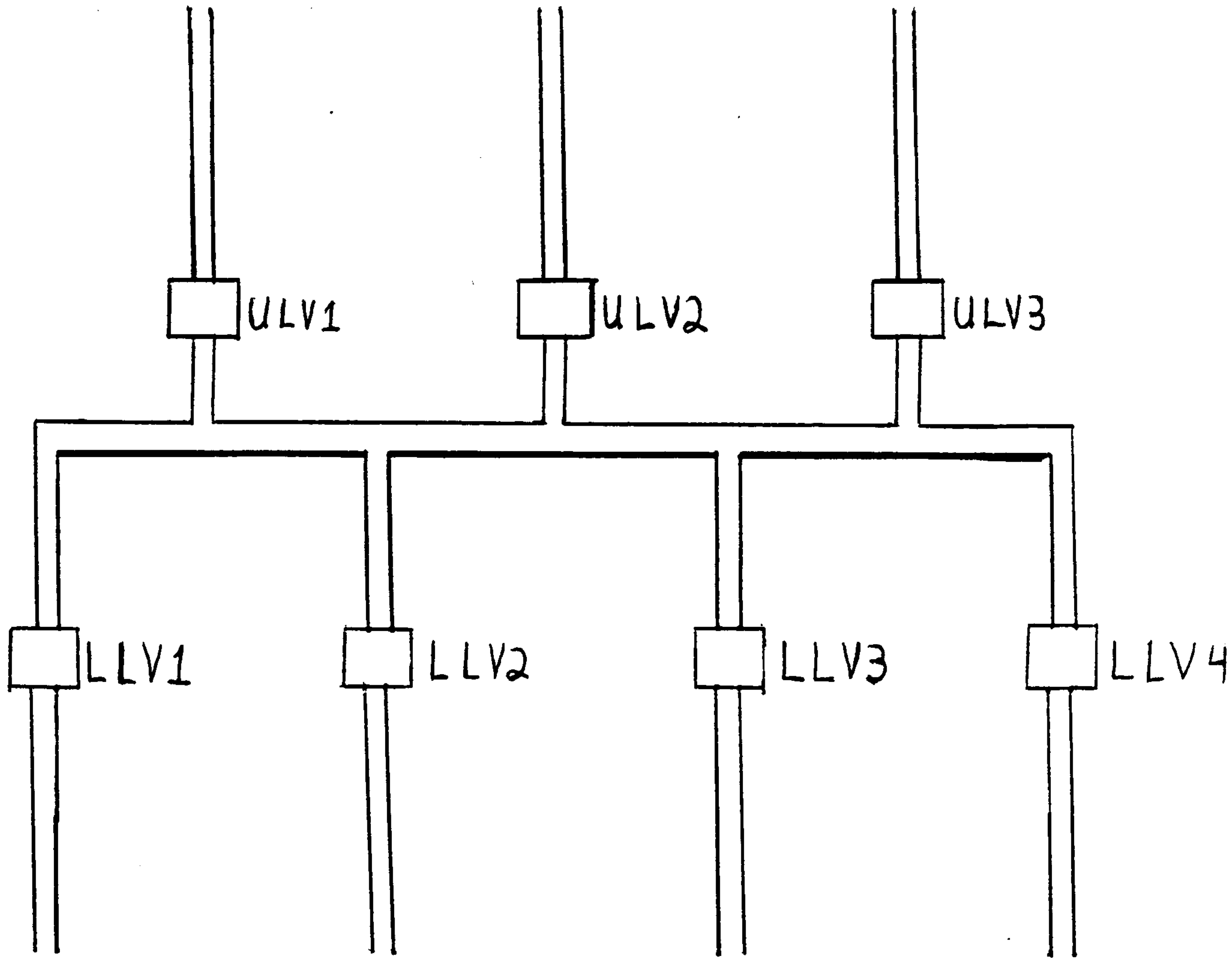
(s) - One or more * - Exactly one

(Configuration Constraint Table)

Tanks, latch valves and engine valves are connected by sections of fuel line. These fuel lines contain line heaters, and have several possibilities for containing failures:

- o Failure of the line heater to turn on, causing the line to freeze
- o Failure of the line heater to shut off, causing the line to overheat
- o A leak in the line, causing loss of fuel and pressure.

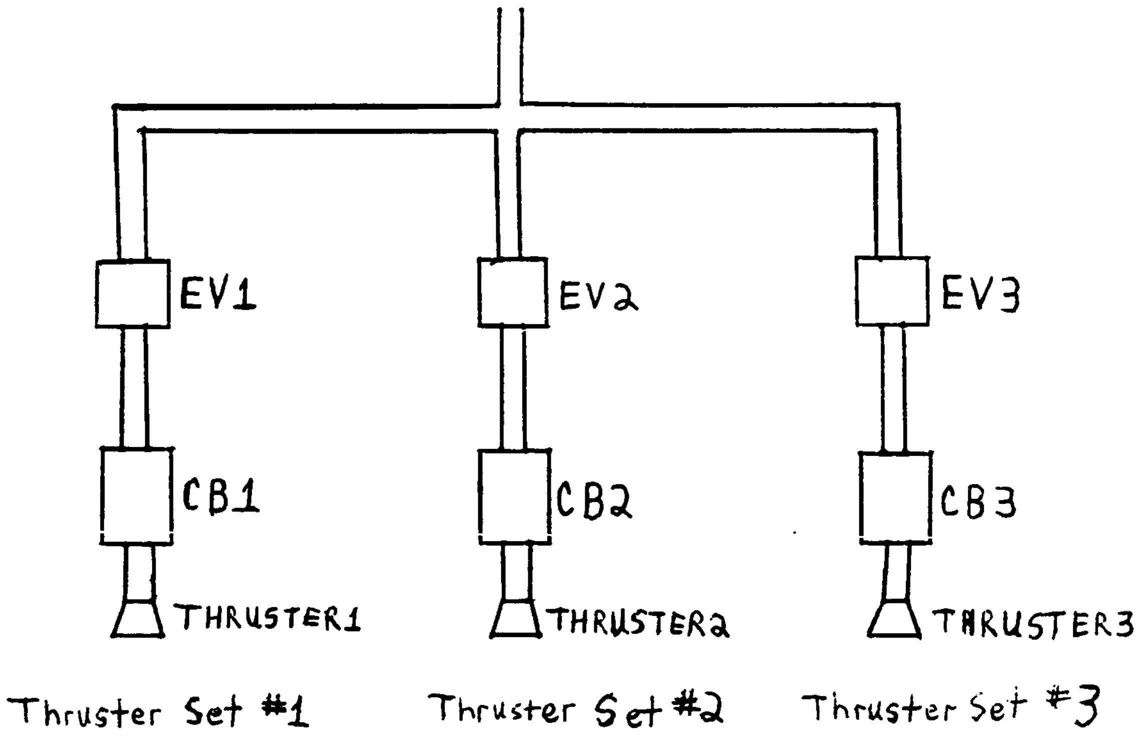
A fuel line is specified by the components at either end of it. It is possible for a section of line to have several components at either end, as shown below:



(Part of the IUE hydrazine propulsion system)

The line between these upper and lower latch valves is functionally only one section of line, with the temperature and pressure constant throughout its length. Any fault within this line will affect all these latch valves. Each upper latch valve (ULV) has below it the components (LLV1, LLV2, LLV3, LLV4). Each lower latch valve (LLV) has above it the components (ULV1, ULV2, ULV3). The line would be specified as the section of line between (ULV1, ULV2, ULV3) and (LLV1, LLV2, LLV3, LLV4).

There are also short sections of fuel line between an engine valve and its catalyst bed, and between the catalyst bed and its thruster. The combination of these three components will be referred to as a thruster set. Each thruster set consists of exactly one engine valve, one catalyst bed and one thruster.

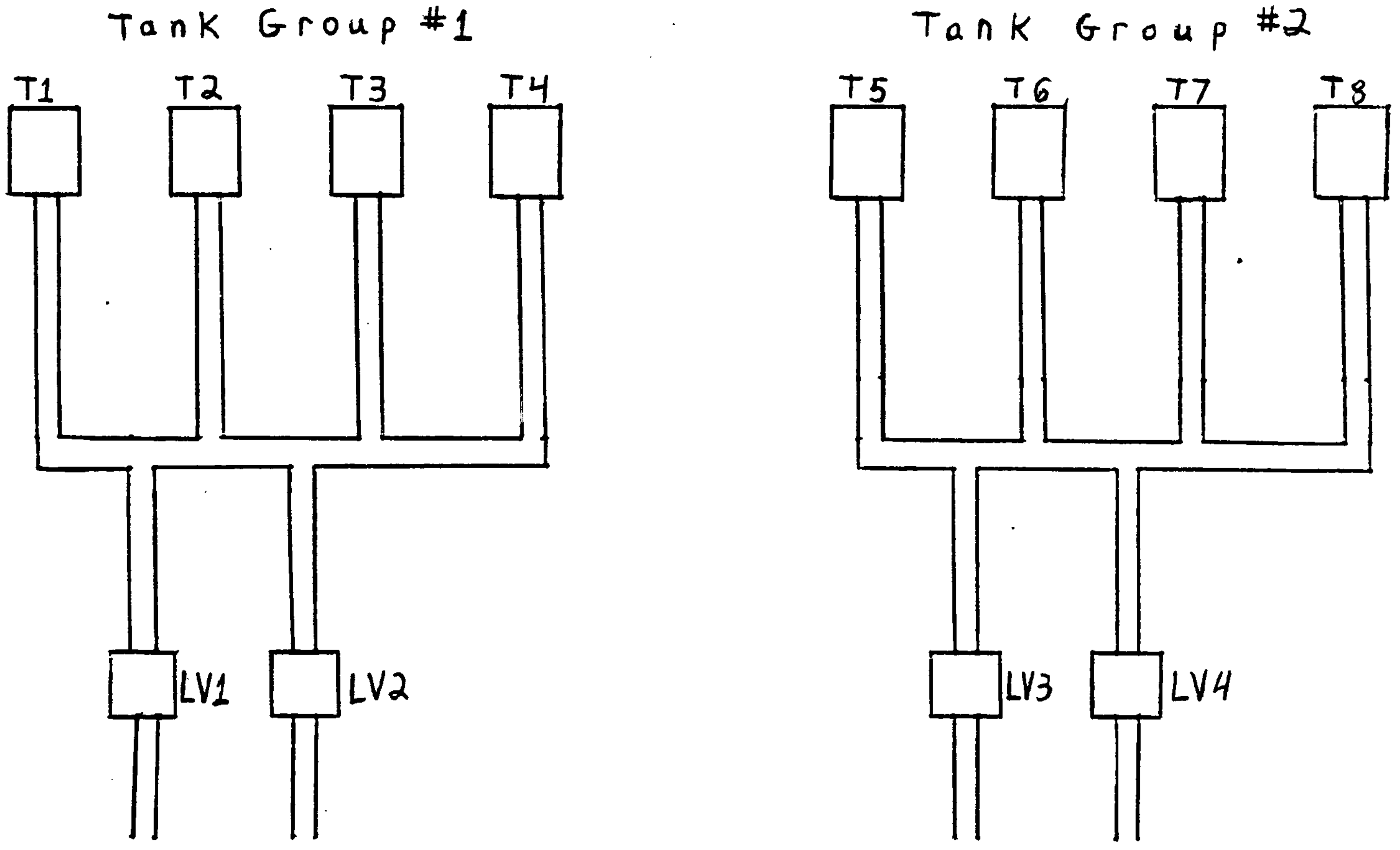


(Part of IUE hydrazine propulsion system)

The expert system does not consider faults for these sections of lines for two reasons:

- The fuel line below an engine valve does not require a heater.
- It is impossible to differentiate leaks below an engine valve from other possible faults (such as a catalyst bed failure).

When considering redundancy, the notion of a tank group is important. An example of tank groups is shown below:



(Part of ISEE hydrazine propulsion system)

Even though there are eight separate tanks, there are only two functional tank groups. Each group has four tanks which are connected via the same fuel line, and have the same group of components directly below them. These are (LV1, LV2) for Tank Group #1 and (LV3, LV4) for Tank Group #2.

A minimum amount of redundancy with regard to tank groups, latch valves and thrusters sets is required for workaround purposes. There must be at least two each of tank groups, latch valves, and thruster sets for a user-configured HPS. This is the minimum amount of redundancy which allows a chance of successful workaround after the fault has been isolated.

The knowledge dealing with the structure of the HPS as described above is maintained explicitly in the internal model of the HPS (see "Software Design Documentation", October, 1984 for a description of the internal model). This knowledge is used by the FHM in both the isolation and workaround stages of the fault handling process. During the isolation process, the fault handler needs to know the structure of fuel lines and valves not only to be able to reason about the initial information it receives when the fault is exposed, but also to be able to "command" specific valves and thrusters when it needs more information. Once the fault has been isolated, the workaround procedures use the structural information to close off parts of the system. This is necessary to prevent, or at least minimize, any effects from the fault in future operation of the system. For example, if a fuel line were leaking, all valves directly above and below that section of line would be closed and considered unusable. This may cause the loss of a set of tanks, a set of thrusters, or a section of fuel line, which is why redundancy is essential in the configuration of an HPS.

<u>Knowledge of the symptoms produced by a fault</u>

This is intended to represent the experience which would be accumulated by a human expert in isolating faults. An abnormally low pressure reading, for example, would usually suggest either a fuel leak or a faulty pressure sensor. This narrows the list of possible faults considerably. A change in attitude which was different than expected, however, could be caused by many things. In this case, a human expert would not be able to deduce much about the fault. The knowledge base is arranged so that the FHM is able to "deduce" the same things as a human, given the same information.

When the FHM is first called, the symptoms which resulted from exposing the fault are made available to it by being put into what is called working memory. After the data representing the symptoms are put into working memory, they will cause one or more rules to "fire". A rule fires when its left hand side (LHS) has been satisfied, which could be thought of as causing the IF part of an IF-THEN clause to become true. When the rule fires, its right hand side (RHS), or the THEN part of an IF-THEN clause, is executed.

The RHS may call procedures which "send" commands to the spacecraft, causing it to change its state and resulting in new data being put into working memory. Also, the RHS may insert data directly into working memory. In either case, the new data may then satisfy the LHS of other rules, which will cause them to fire. This cycle will continue until some goal has been achieved, such as isolating a fault. One possibility for rules would be:

<u>Rule #1</u>

IF: A catalyst bed sensor reports an abnormal temperature

THEN: Assert catalyst-bed-heater-or-sensor-fault AND

Fire the thruster below that catalyst bed

<u>Rule #2</u>

IF: There is catalyst-bed-heater-or-sensor-fault AND

There is unexpected attitude adjustment

THEN: Assert catalyst-bed-heater-fault

If this were part of an expert system, something might be placed in working memory saying there was a symptom of an abnormal temperature reading from a catalyst bed sensor. Rule #1 would then fire and place in working memory an assertion that there was a catalyst bed heater or sensor fault, and request that the thruster below the catalyst bed be fired. If the fault was with the catalyst bed heater as opposed to the temperature sensor, then the attitude adjustment which resulted from firing the thruster would be different than expected. Rule #2 would then fire and assert that there was a catalyst bed heater fault.

The rules in the FHM are written and ordered such that at least one rule will fire as soon as there are any symptoms, and any rule that initially fires will cause a significant reduction in the number of possible faults under consideration. This is possible because each "class" of faults (such as a fuel leak, stuck valve, broken heater, etc.) has a symptom or small combination of symptoms that are unique to that class. By writing rules which look for exactly those symptoms, initially reducing the fault possibilities becomes straightforward.

Consider the symptom of an abnormally low pressure reading, for example. There are several faults which could cause this symptom. It could be a faulty pressure sensor, a fuel line leak, or an engine valve stuck open. However, in addition to the rule which checks for abnormal pressure, there is a rule which checks for an engine valve being stuck. If that rule hasn't fired when the abnormal pressure rule fires, we know that the fault is not an engine valve stuck open, therefore it must be a faulty sensor or line leak.

Precise location of the fault is not as simple as the initial reduction of possibilities. The fault can sometimes be pinpointed fairly well by reasoning with the initial symptoms, but other cases may require various actions to be performed to obtained additional data. Use of these actions is discussed in the next section.

Knowledge about what symptoms may be produced by a fault is implicitly stored, as opposed to the explicit storing of the structural knowledge. This knowledge is imbedded in both the actual rules and in the firing sequence of the rules. In the actual rules, the knowledge is used in writing the set of symptoms and other data necessary for the rule to fire. It is also used in deciding what actions the rule should request, since actions are geared to uncovering specific symptoms. In the firing sequence of the rules, knowledge about the symptoms of a fault is useful when determining which rule to fire when the same set of symptoms could cause two or more rules to fire. If a system chose randomly between two rules which could fire, it might choose one that was only useful when isolating a fault which in this case had been ruled out. This is important to consider, since firing a rule usually changes what data is in working memory. Properly using knowledge about the symptoms of a fault will cause certain rules to fire in a particular order, analogous to the way in which experts would try out their hypotheses.

Knowledge of how an action should affect the readings from a sensor

Knowing the effect an action should have on sensor readings under both normal and faulty operating conditions corresponds to another type of knowledge acquired by a human expert. When trying to isolate a fault, various actions can be performed to provide additional information. The resulting information would be useless unless the expert knew what the results of each action should be if all affected components were functioning properly, and had some idea of what the readings might be if a suspected component were indeed malfunctioning. This knowledge in the FHM enables the choosing of an action which will generate the most useful data for the isolation process.

When designing a rule base, points will be found in a rule sequence where additional data is necessary for the system to proceed. At those points, the RHS of a rule will request an action to be performed. Performing the action may cause other symptoms, which would then fire other rules. The action chosen will be the one most likely to yield the data needed to proceed in the fault isolation process.

Examples of actions which could be requested by the FHM are the closing of a latch valve or the firing of a thruster. A latch valve might be closed to check a section of fuel line. If a symptom persists after closing the valve, then the fault which is causing that symptom lies somewhere between the closed valve and the sensor. If the symptom does not persist after closing the valve, the fault lies on the other side of the valve from the sensor.

Firing a thruster is used mainly to determine whether or not the fault is affecting the attitude adjustment of the spacecraft. Nearly all component failures will affect the attitude adjustment in some manner, while a faulty sensor will have no effect. Therefore, firing a thruster is used in nearly every rule sequence to discriminate between component and sensor failures.

Knowledge about the effect of an action on the readings from a sensor is, like the knowledge about fault symptoms, implicitly stored. This knowledge is embedded mainly in the rules. As mentioned above, knowledge about the effects of actions is useful when determining which action should be performed. Also, knowing what symptoms an action may uncover is useful when determining what symptoms are required for the next rule in a sequence.

None of these types of knowledge are totally distinct from each other. Structural knowledge is used in determining how actions will affect a sensor reading, and also in determining the symptoms of a fault. Also, the purpose of understanding the effects of an action is to interpret the symptoms resulting from an action, so both this knowledge and the knowledge about the symptoms of a particular fault are taken into account in writing the rules of a system.

Example of FHM isolating fault

Fault: fuel line leak below latch valve #1 (in the ISEE configuration)

Symptoms: Pressure sensors indicate abnormally low pressure

Rule sequence:

#1 PRESSURE-ABNORMAL rule fires

(Deduction: Fault is fuel leak or faulty pressure sensor)

ACTION: Fire-random-thruster

(Reason: Get more data, causing more rules to fire)

#2 ATTITUDE-ABNORMAL rule fires

(Deduction: Fault affects the attitude adjustment when a thruster is fired)

#3 FUEL-LEAK rule fires

(Deduction: There is a fuel leak in either a tank or fuel line)

ACTION: Close-all-valves

(Reason: Help isolate tank or line section containing leak)

ACTION: Open-leftmost-topmost-latch-valve

(Reason: Check section of line immediately below that latch valve)

#4 PRESSURE-ABNORMAL rule fires

(Deduction: Fault has been found. There is a leak in the fuel line between the latch valve which was just opened and the component or components directly below it)

ACTION: Workaround-line-leak-fault

(Reason: Will close off that section of line, system will work as normal with redundant line and thrusters used)

#5 SYSTEM-WORKAROUND-DONE rule fires

(Deduction: The FHM is finished with isolation and workaround of fault. The hydrazine system should now be functional, relying on redundant componentry and pathways)